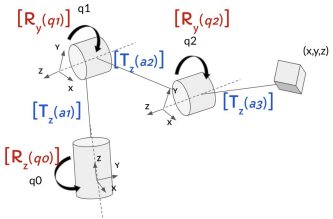
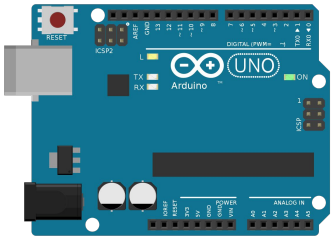
{ Tu primera fábrica automatizada; }

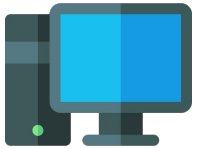
Visión Computacional

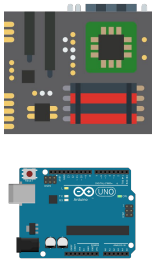
{ *hoy nos damos **color*** 🌈🌈 }

¿Qué hemos visto hasta ahora?



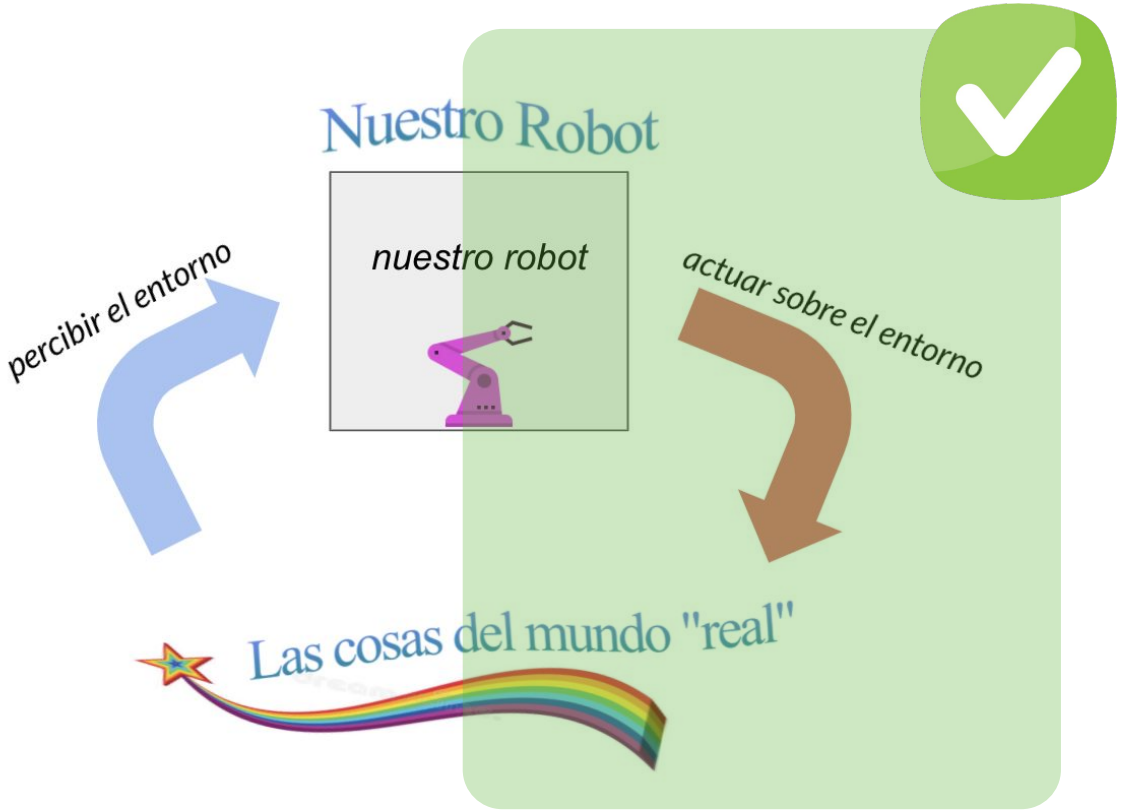






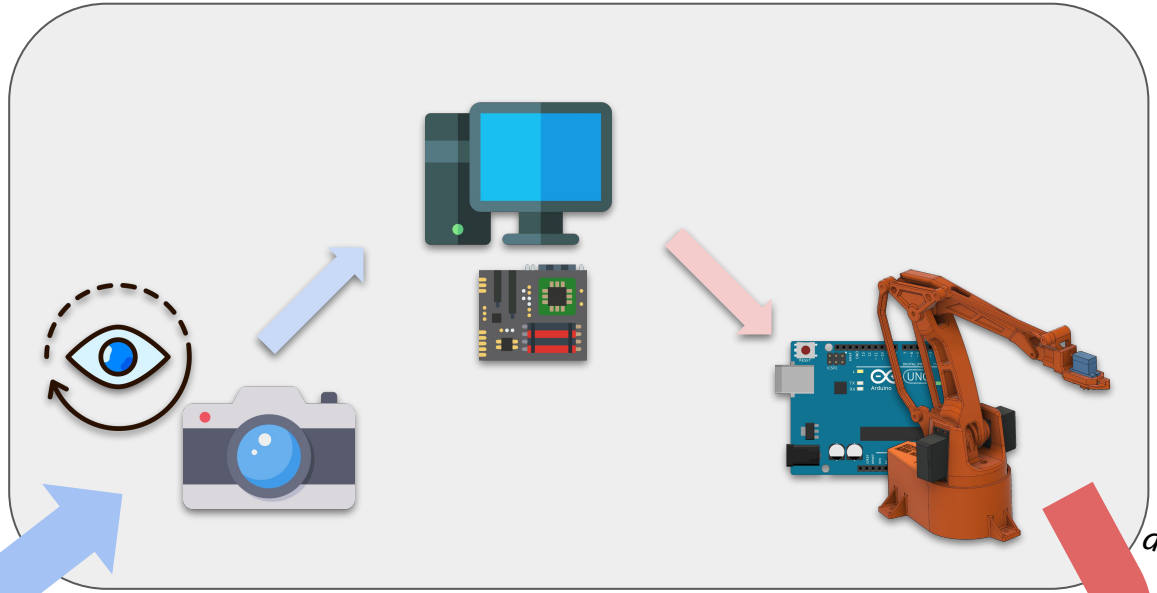
```
#include <Servo.h>

#define pin_servo 10
#define pin_boton1 4
#define pin_boton2 5

Servo servolindo;
int ang = 0;

void setup() {
  pinMode(pin_boton1, INPUT);
  pinMode(pin_boton2, INPUT);
  servolindo.attach(pin_servo);
  Serial.begin(9600);
}

void loop(){
  if (digitalRead(pin_boton1)){
    ang += 5;
  }
  if (digitalRead(pin_boton2)){
    ang -= 5;
  }
  Serial.println(ang);
  servolindo.write(ang);
  delay(100);
}
```

Pero recordemos la idea fundamental



Los robocs de hoy en día perciben de manera más compleja

Necesitamos agrandar un poco el esquema



percibir el entorno

actuar sobre el entorno

{

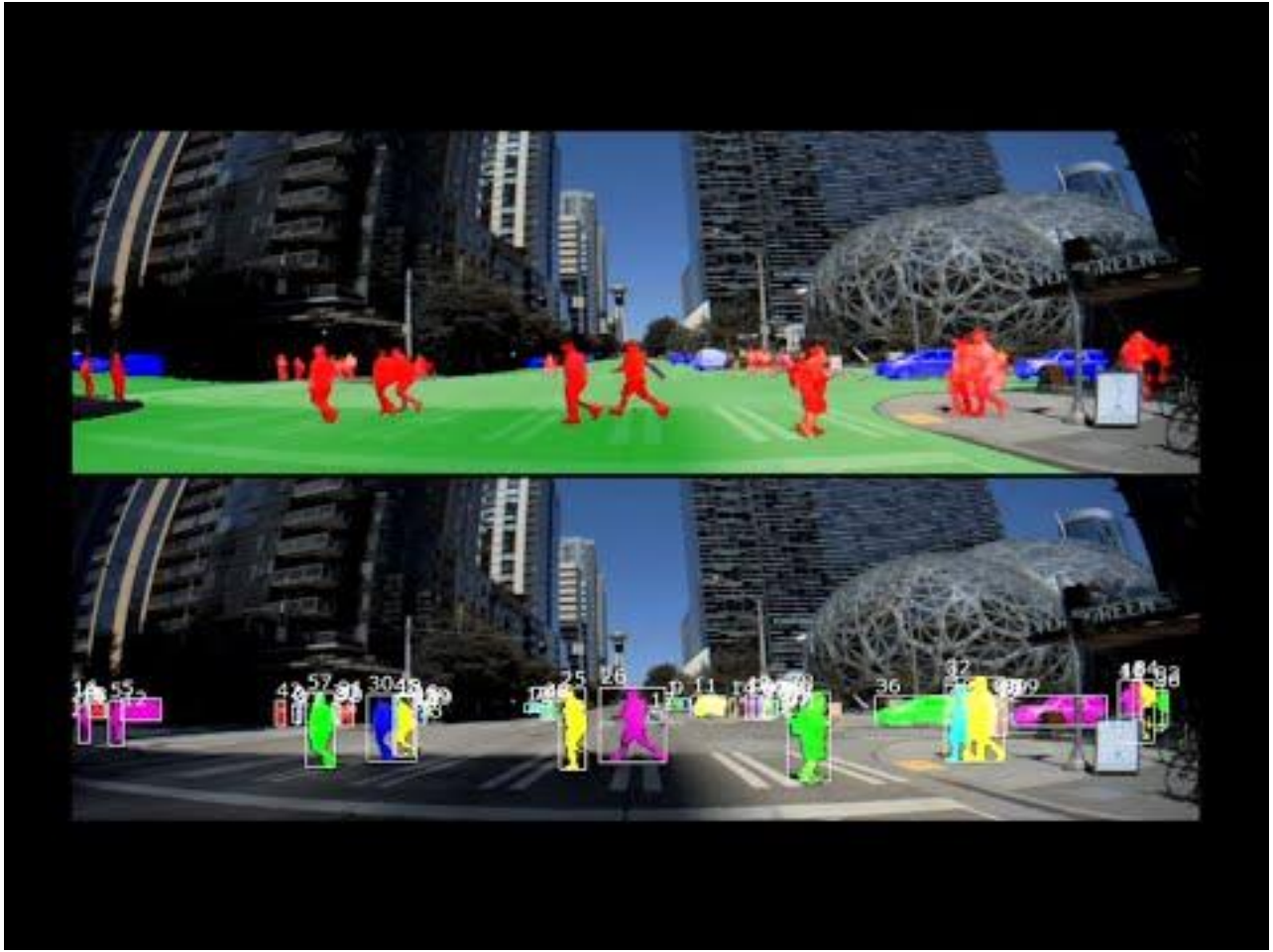
Procesamiento de Imágenes y Visión Computacional

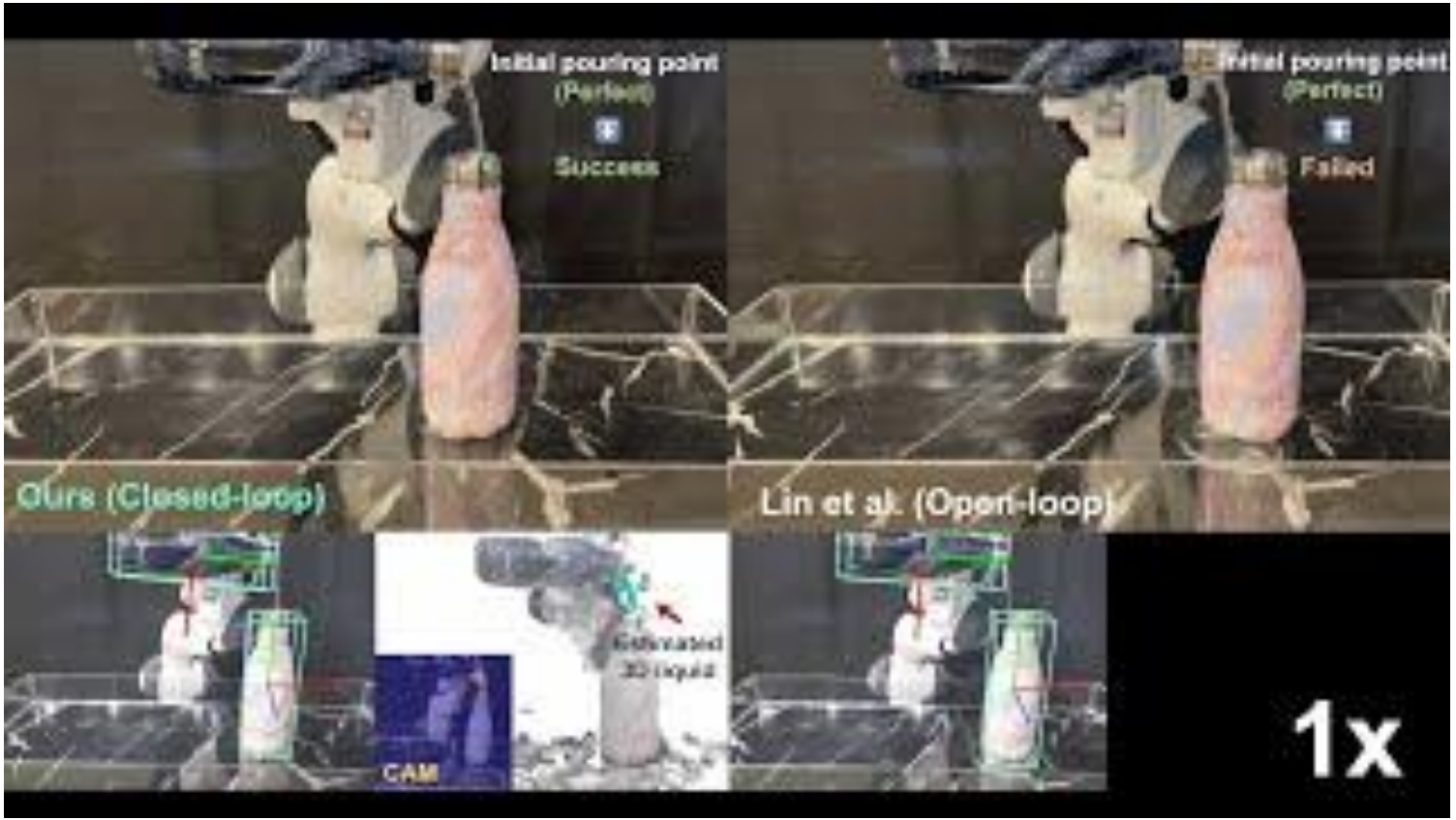
}

{

MOTIVACIÓN

}



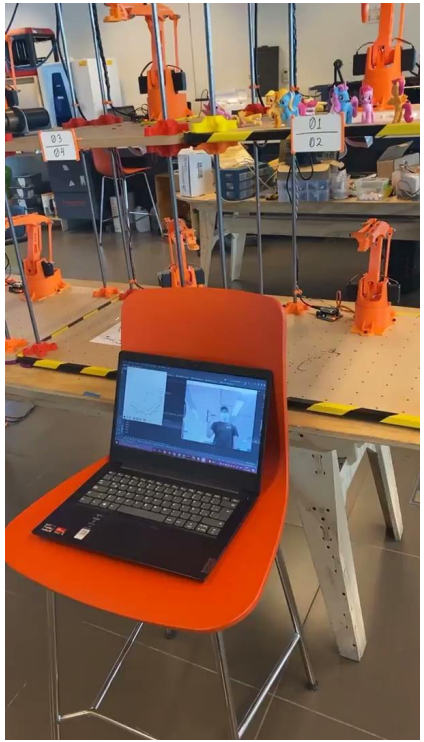

Initial pouring point
(Perfect)
Success
Initial pouring point
(Perfect)
Failed
Ours (Closed-loop)
Lin et al. (Open-loop)
CAM
1x

Aplicaciones en My Little Factory

- Detección de manos
- Detección de formas geométricas (puzzle)
- Detección de engranajes

Detección de manos (sus auxs)

1. Identifica puntos clave de la mano
2. Interpreta el gesto correspondiente
3. Manda el comando necesario

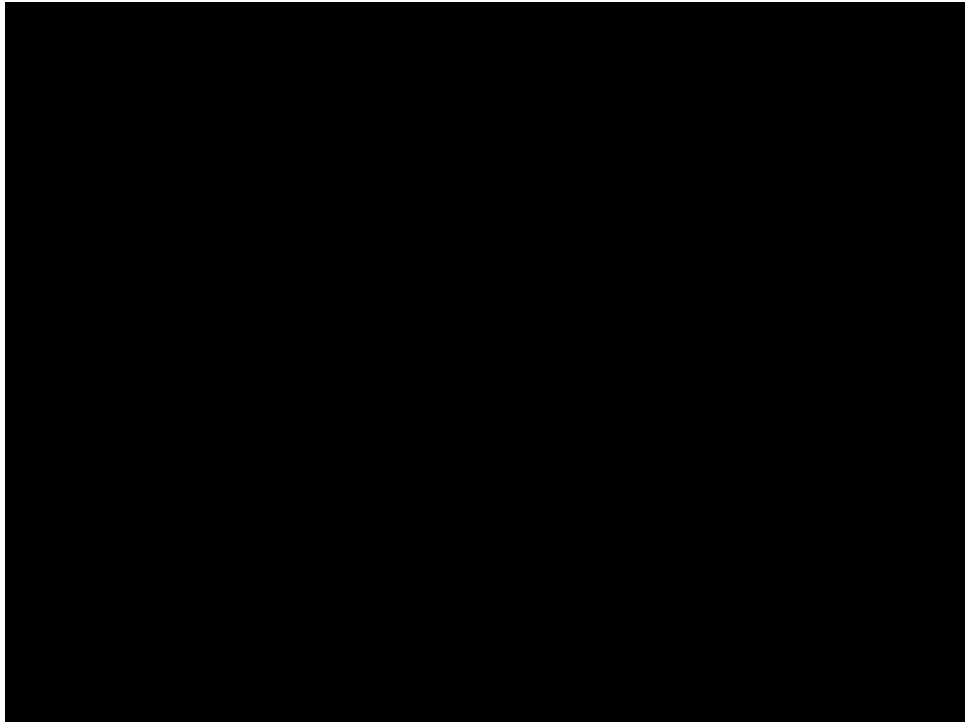


Detección de formas (puzzle)

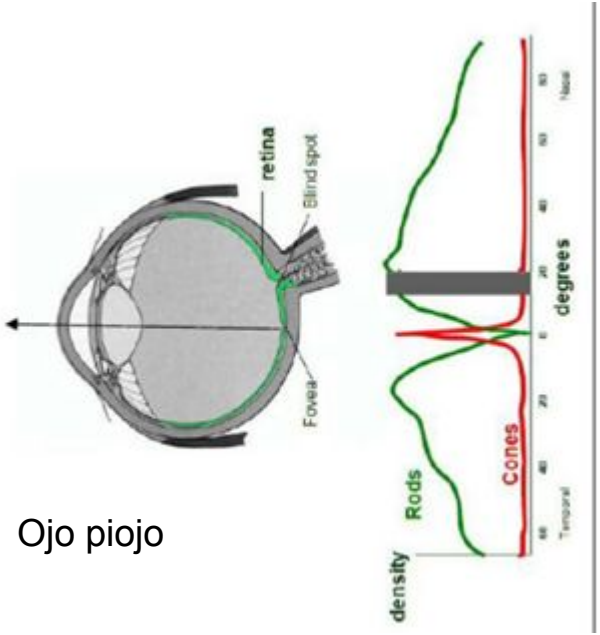
1. Detectar figura
2. Contar bordes e identificar figura
3. Encontrar posición
4. Enviar instrucciones al robot

Detector y selección de engranajes

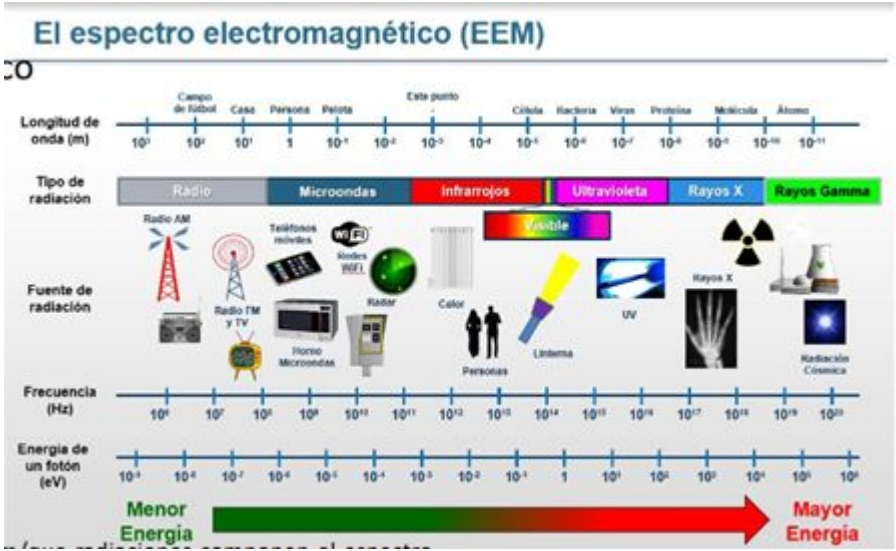
1. Identifica el engranaje
2. Extrae el número de dientes y la posición
3. Envía instrucciones al brazo, separando por número de dientes



Nuestra visión:

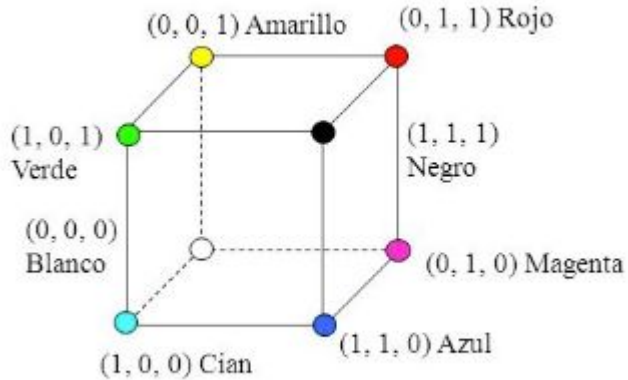


Ojo piojo



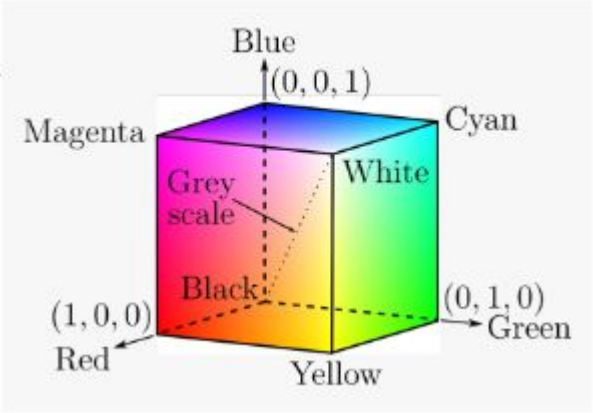
Espacios de colores

- CMYK (cyan magenta yellow black)
- Modelo sustractivo
- Impresoras



Espacios de colores

- RGB (Red, Green, Blue)



24 bits (8 bit o 256 niveles de color por canal)
256×256×256 ≈ 16,7 millones de colores

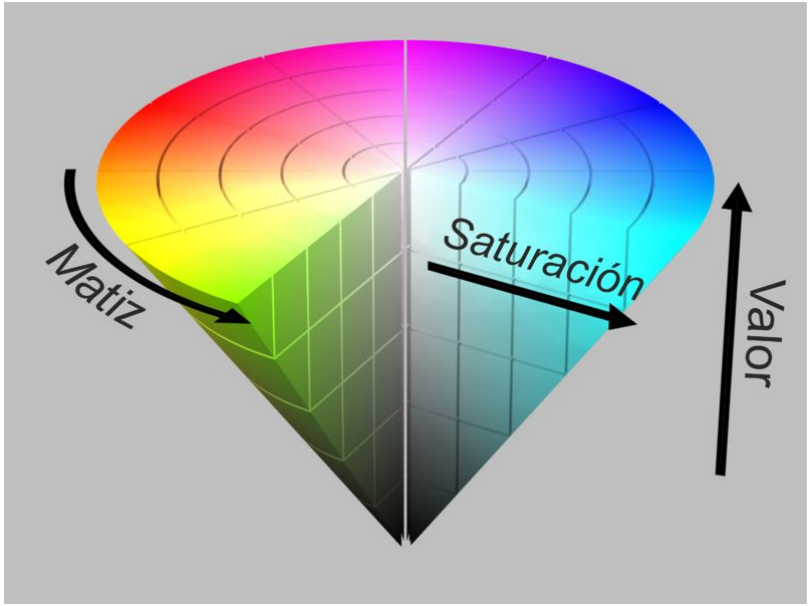
- (0, 0, 0) es negro
- (255, 255, 255) es blanco
- (255, 0, 0) es rojo
- (0, 255, 0) es verde
- (0, 0, 255) es azul
- (255, 255, 0) es amarillo

Espacios de colores

- HSV

  - Saturación -> Brillo (0 a 100%)

  - Valor - > Luminosidad (0 a 100%)



¿Qué es una imagen?

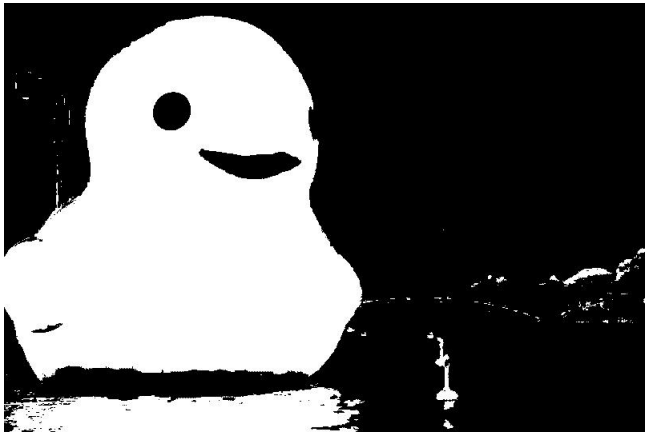
¿Qué es una imagen (digital)?

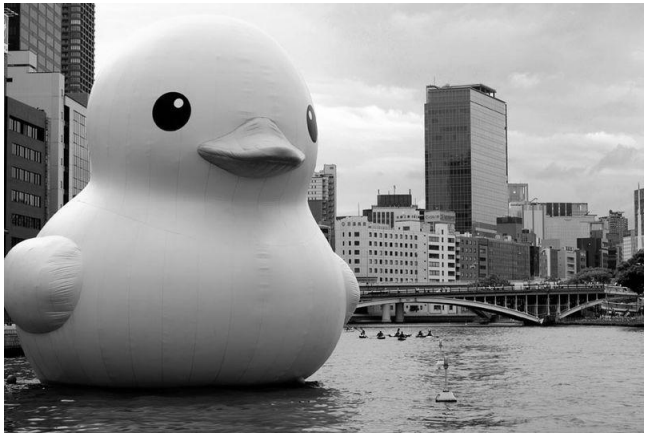
- Es una colección ordenada de píxeles, como una **matriz**
- Donde cada pixel es un elemento de la matriz
- Cada elemento es el color que tendrá cada pixel

¿Y un video?

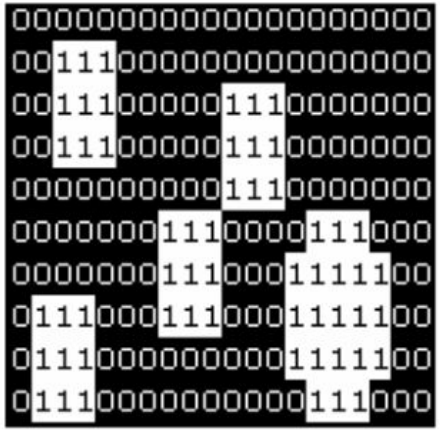
Tipos de imágenes

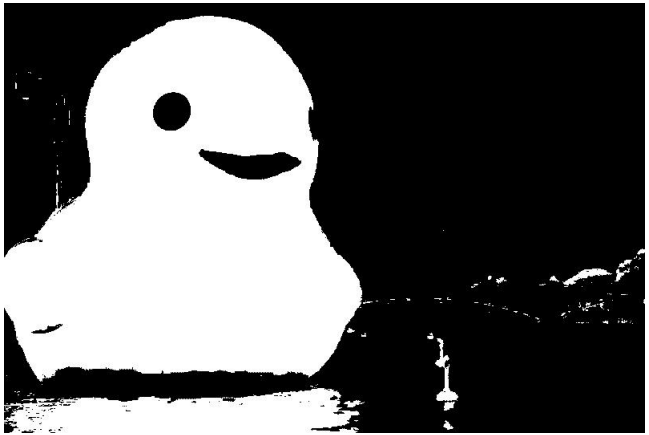
- Binaria
- Escala de grises
- Color



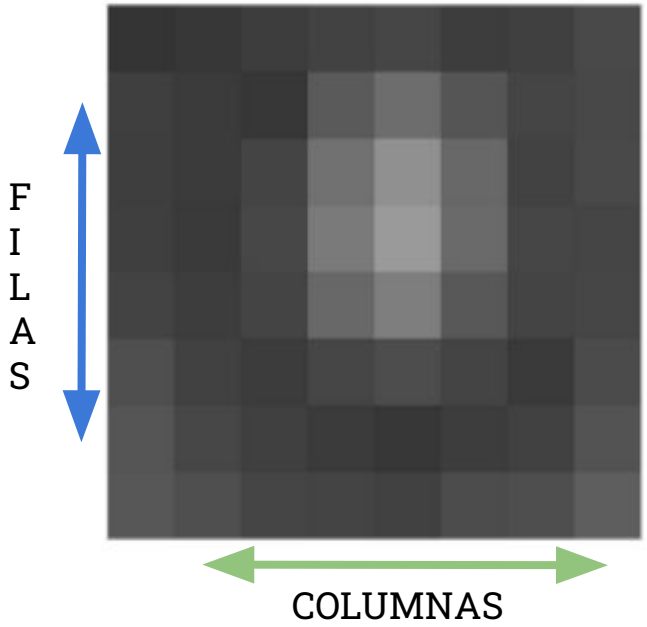


¿Qué es una imagen binaria?





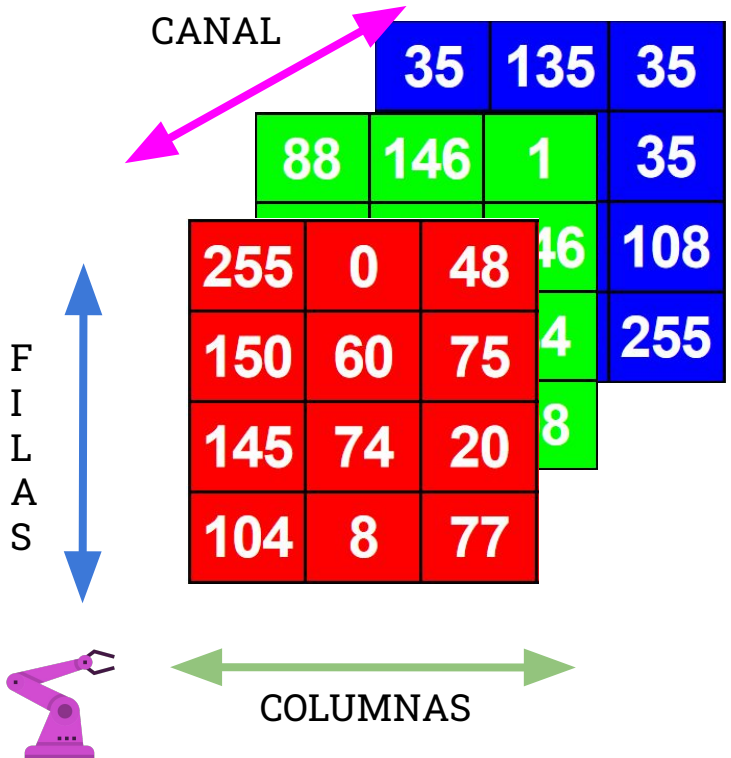
¿Qué es una imagen en escala de grises?



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

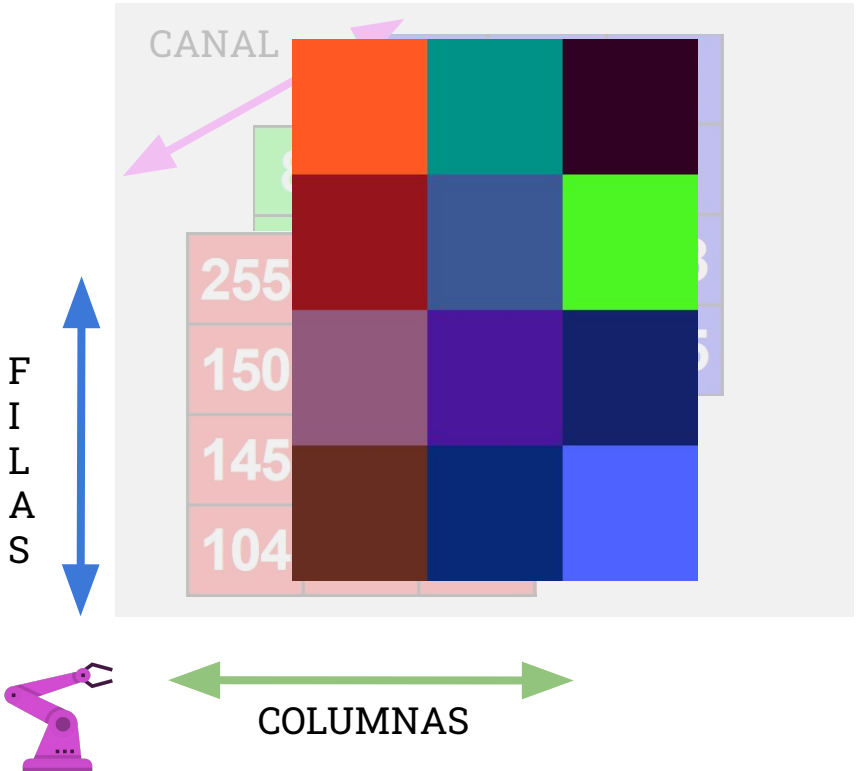
En una imagen regular, los valores van de 0 a 255, más oscuro a más claro

¿Cómo sería la matriz de una imagen RGB?




```
[
[[255,88,35], [0,146,135],  [48,1,35]      ],
[[150,20,28], [60,88,148],  [75,246,35]   ],
[[145,89,124], [74,22,155],      [20,34,108]   ],
[[104, 45, 33], [8,41,120],   [77,98,255]   ]
]
```

¿Cómo sería la matriz de una imagen RGB?




```
[
[[255,88,35], [0,146,135],  [48,1,35]      ],
[[150,20,28], [60,88,148],  [75,246,35]    ],
[[145,89,124], [74,22,155], [20,34,108]    ],
[[104, 45, 33], [8,41,120],   [77,98,255]  ]
]
```

*Como son matrices, para un computador es re-fácil**

*Hasta cierto nivel

OpenCV

- Librería de visión por computador de código abierto
- Existe gran cantidad de documentación y ejemplos
- Soporta lenguajes : C, C++, **Python**, Ruby, Matlab
- Compatible con Linux, Windows y Mac OS X.

Qué nos permite OpenCV

- Interpretar una imagen (matriz)
  - Extraer información de sus canales (espacios de color)
- Operar:
  - Operaciones Aritméticas: Suma, Resta, Ponderación, Suma Ponderada
  - Operaciones Lógicas: AND, OR, XOR, NOT
  - Umbralización Binarización, Truncado, Umbral a cero, Inversos
- Implementar: Algoritmos de reconocimiento

Suma o resta de imágenes

```
sub = cv2.subtract(image1, image2)
add = cv2.add(image1, image2)
```

image1

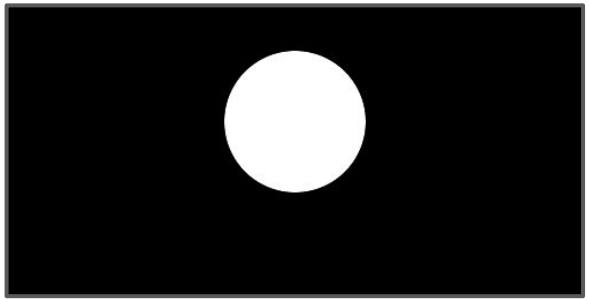
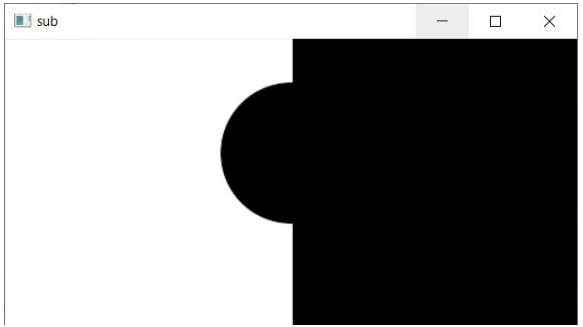
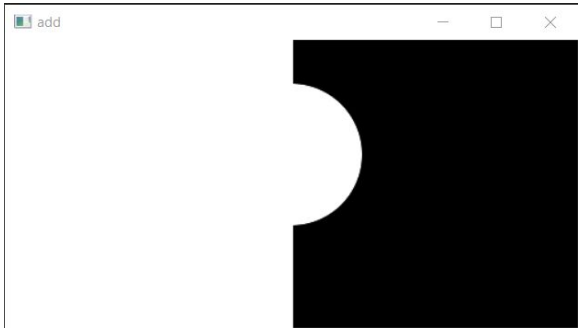


image2





Operador OR y AND

```
or_ = cv2.bitwise_or(image1, image2)
and_ = cv2.bitwise_and(image1, image2)
```

image1

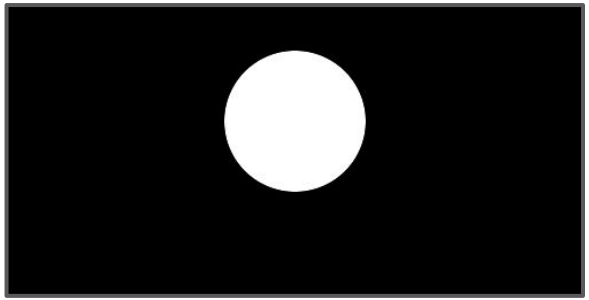
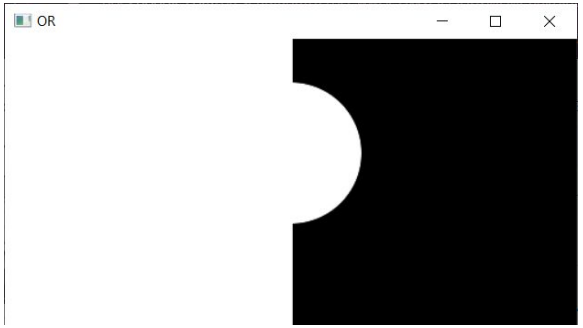
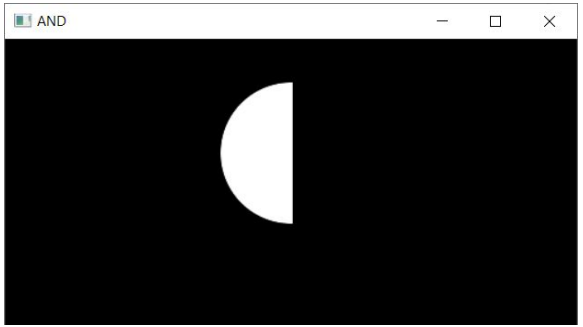


image2



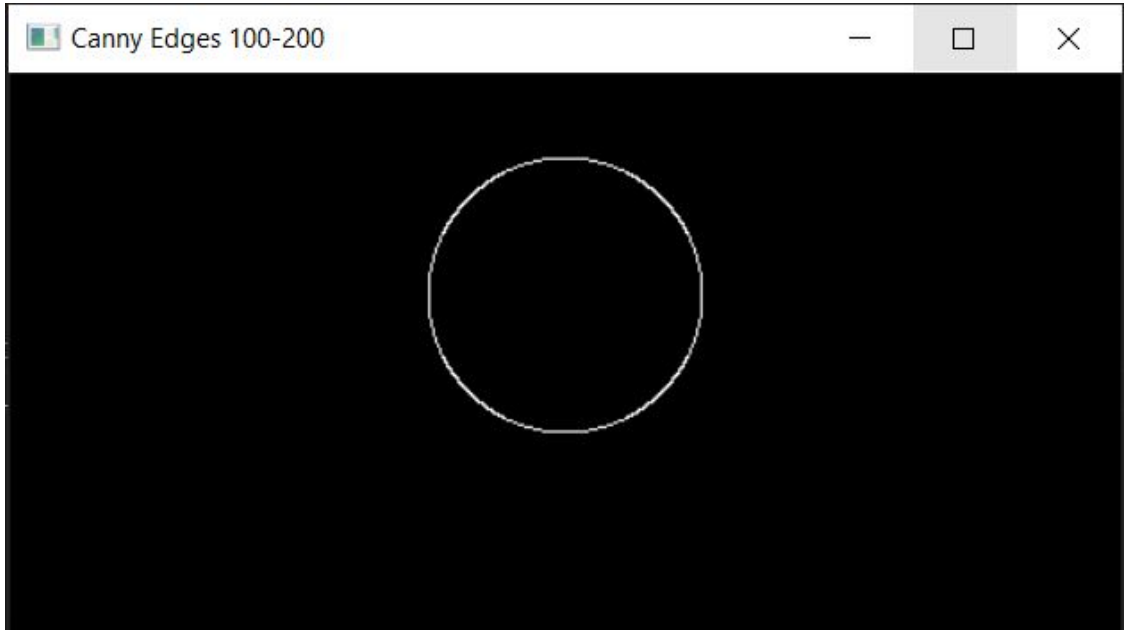


Operaciones Básicas

- Máscara

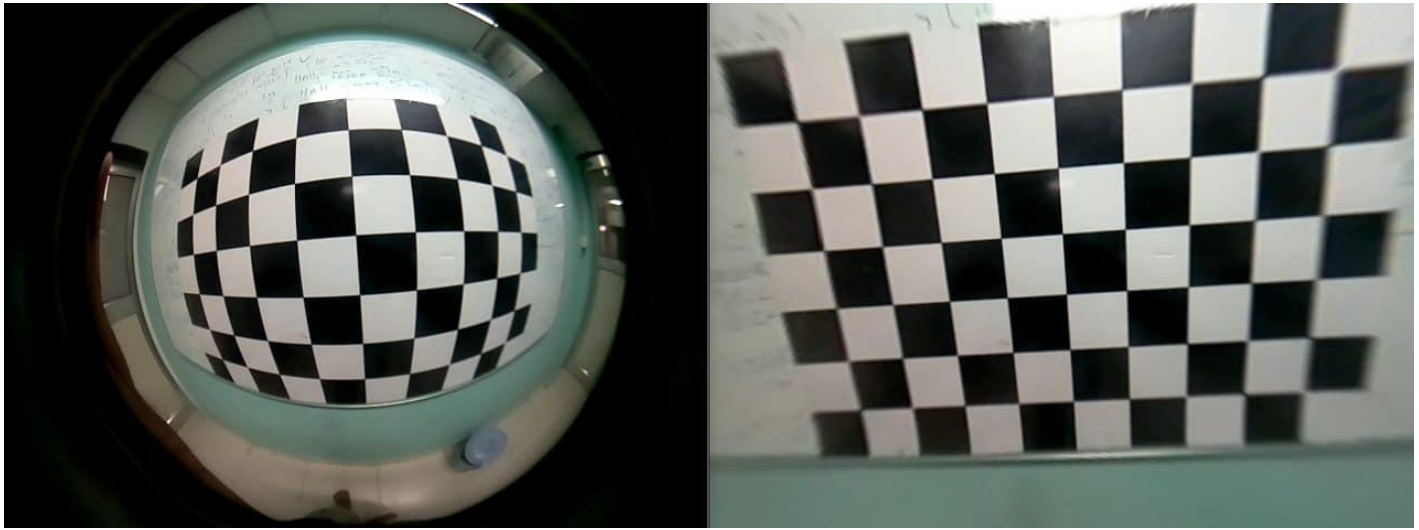
Detector de bordes

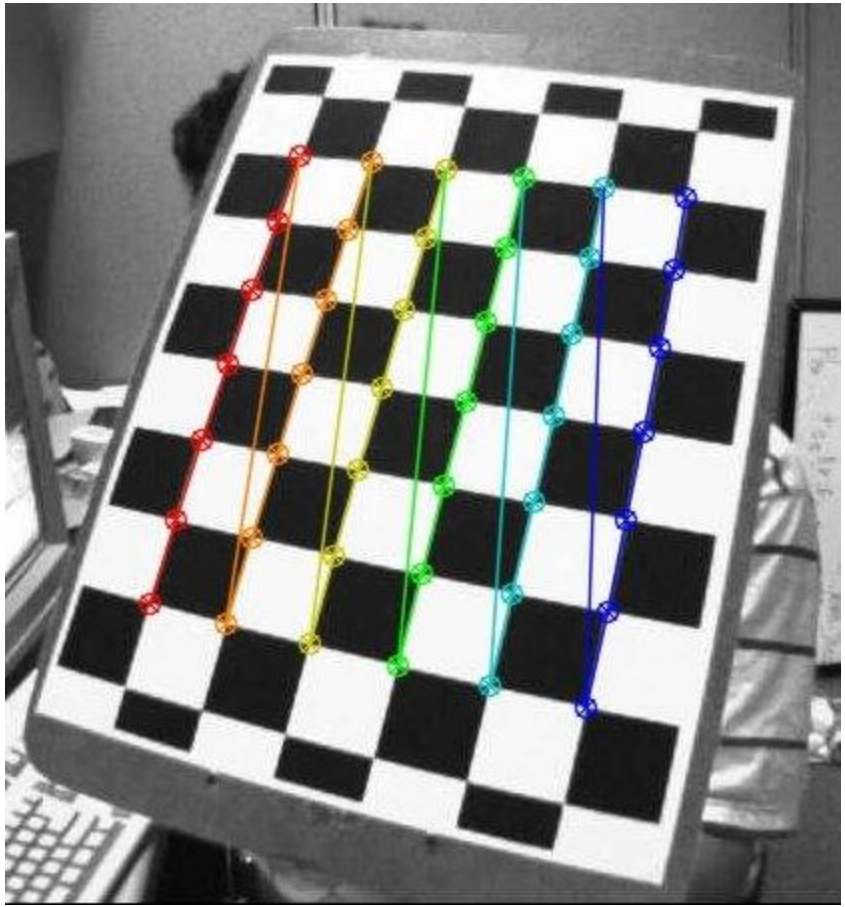
```
lower_thr = 100
upper_thr = 200

edges = cv2.Canny(image2,
        lower_thr, upper_thr)
```



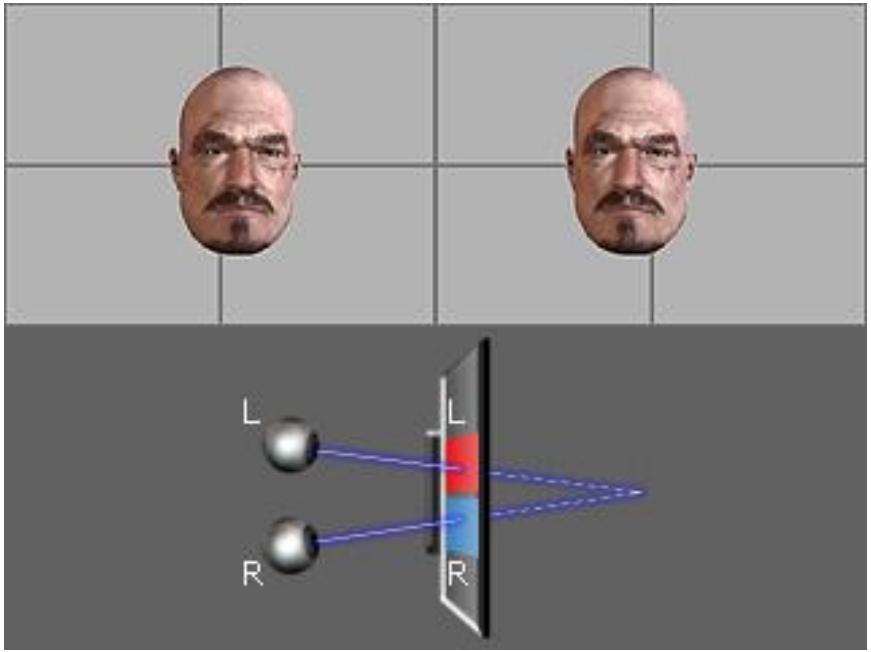
Calibración Cámara

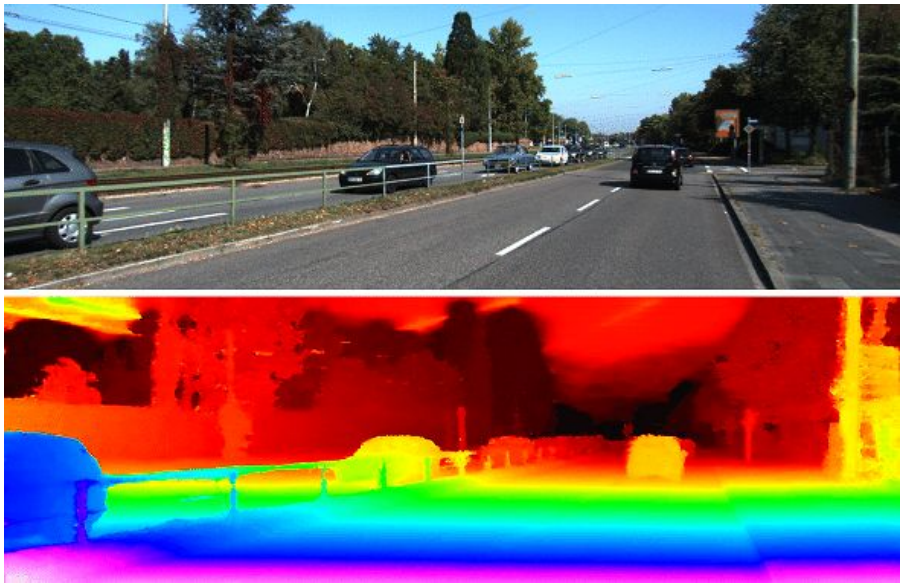
Problema:



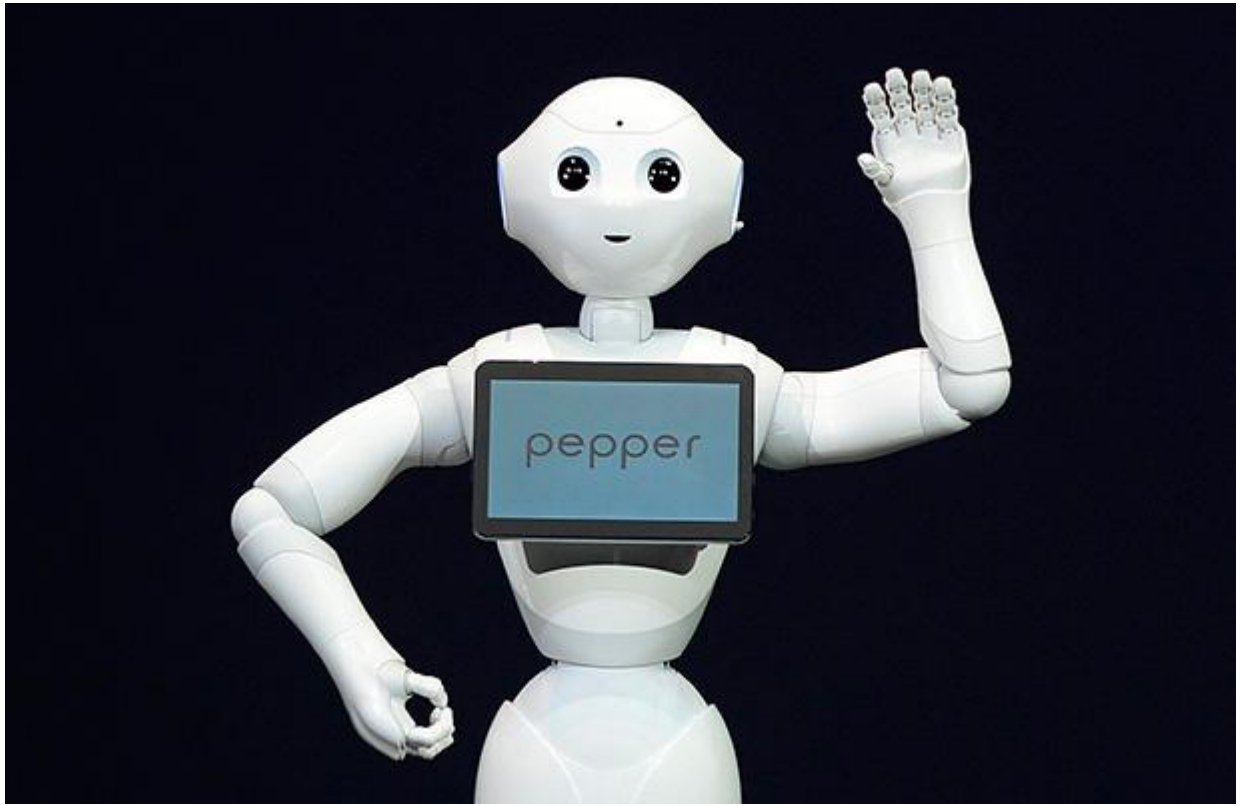


Vision stereo



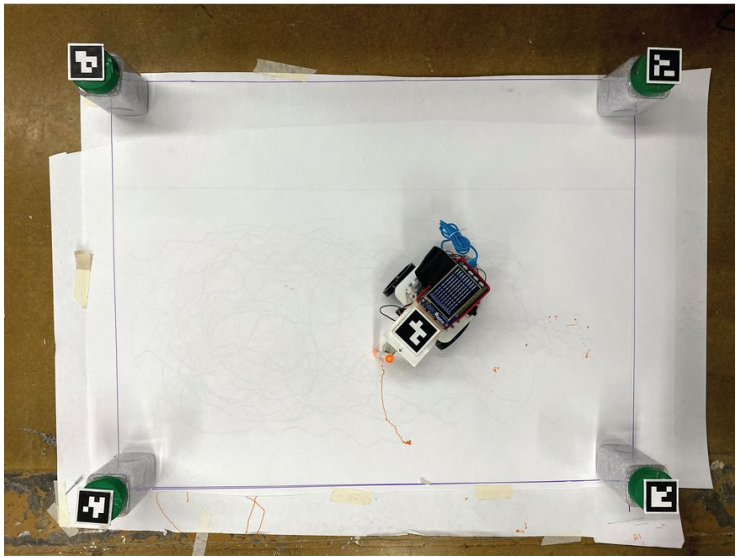


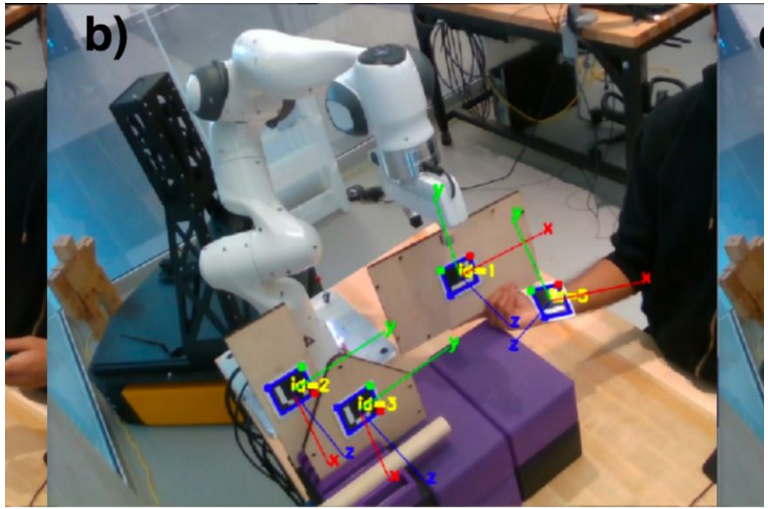
Ejemplo: (Lo conocieron la semana pasada)



Estimación de posición

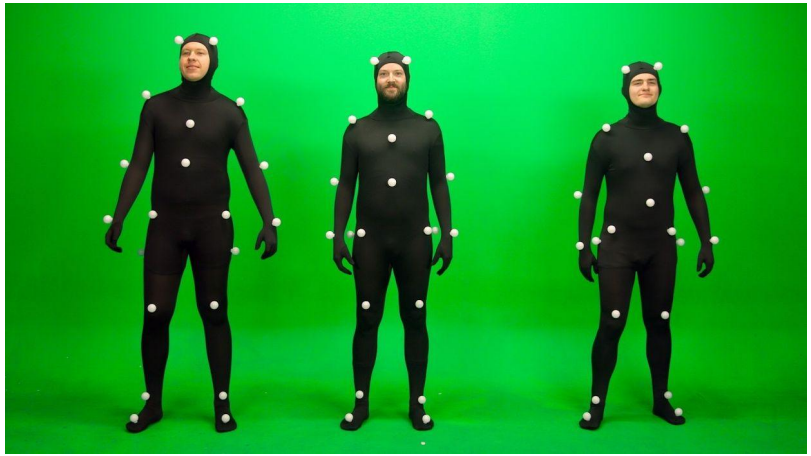
Aruco Markers:

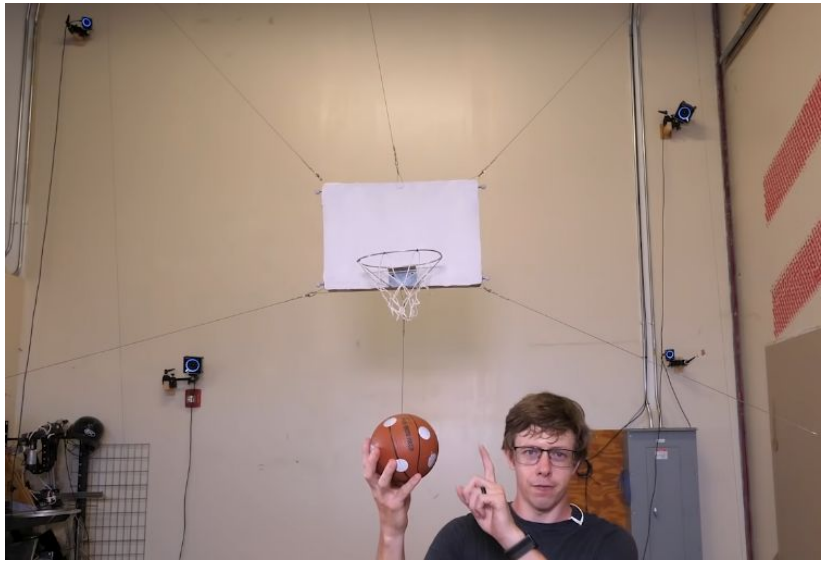




Estimación de posición

Motion capture:





Redes Neuronales

YOLO: You Only Look Once

Ejemplo concreto:

Minería: Estimación de duración de operaciones de minería

Zippedi

Ahora a Python

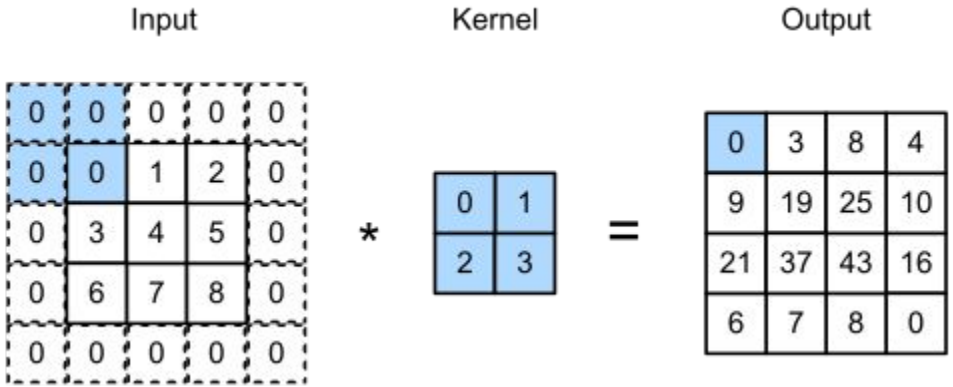
La librería que usamos se llama OpenCV

```
import cv2

# Cargar imagen
image = cv2.imread("carpeta/image.jpg")

# De lista a np.array
mat = np.array(listas_de_listas)

# Mostrar imagen
cv2.imshow('nombre_ventana', image)
cv2.imshow('nombre_ventana2', mat)
```

```
[
[[255,88,35], [0,146,135],  [48,1,35]      ],
[[150,20,28], [60,88,148],  [75,246,35]   ],
[[145,89,124], [74,22,155],      [20,34,108]   ],
[[104, 45, 33], [8,41,120],   [77,98,255]   ]
]
```

Operaciones no tan Básicas

- Filtros Convolucionales

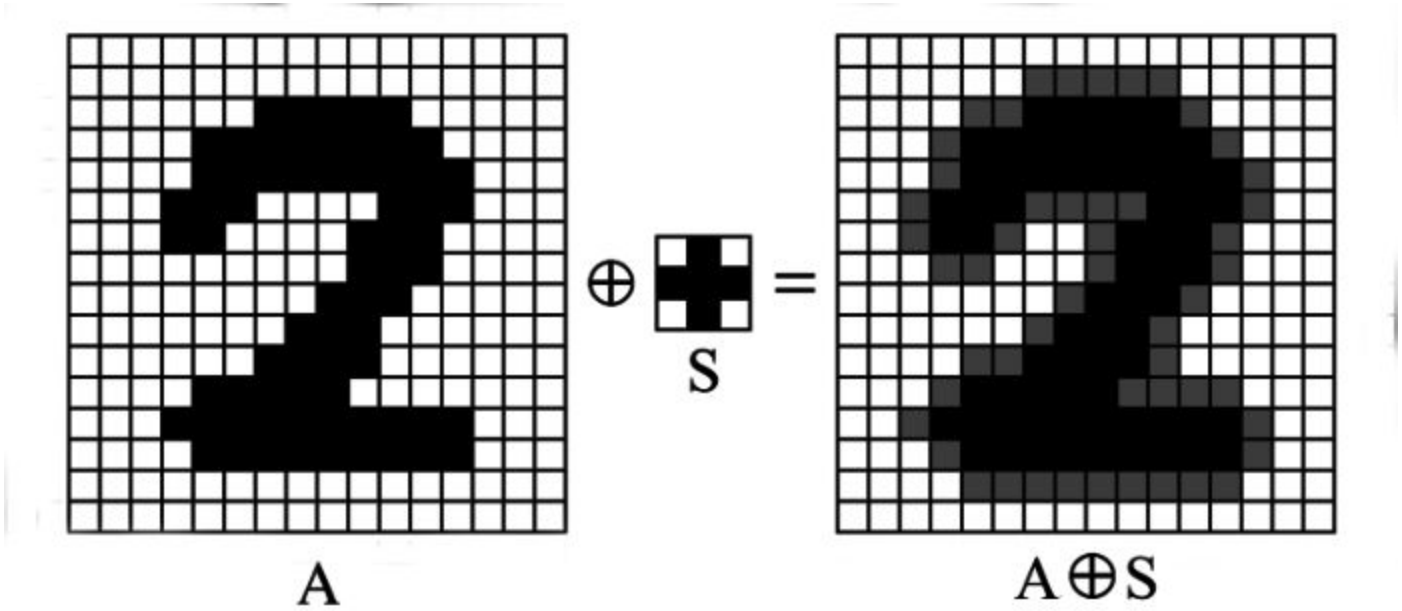


Tipos de Filtros

- Promedio $$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
- Gaussiano $$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$
- Mediano

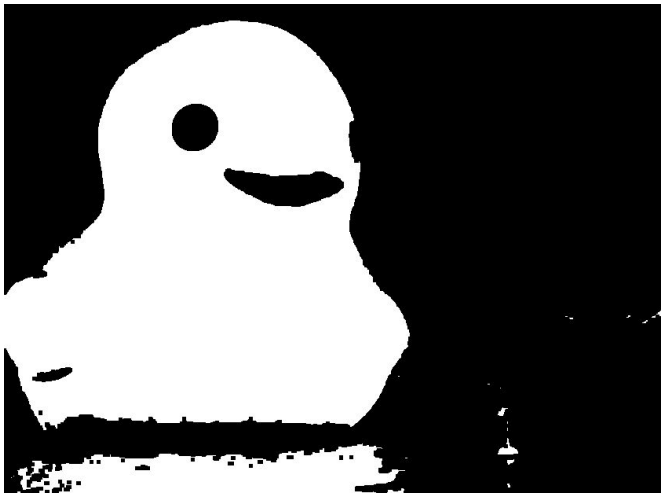
Operaciones morfológicas

Dilate



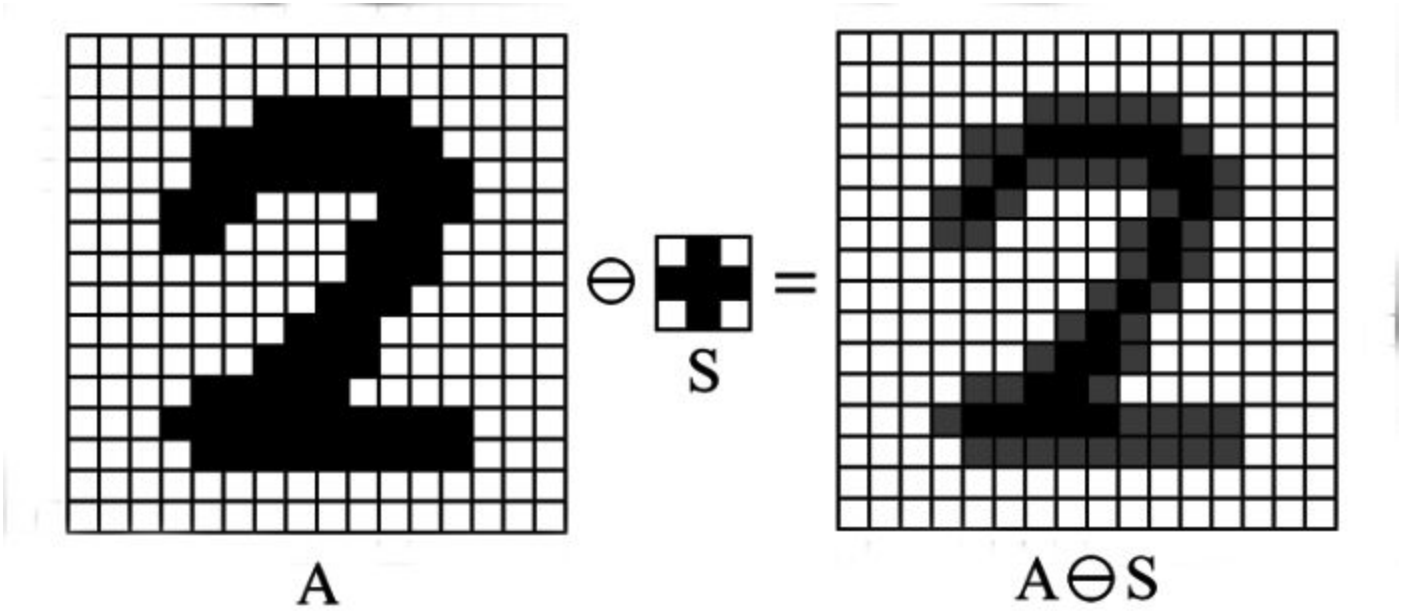
Operaciones morfológicas

Dilate



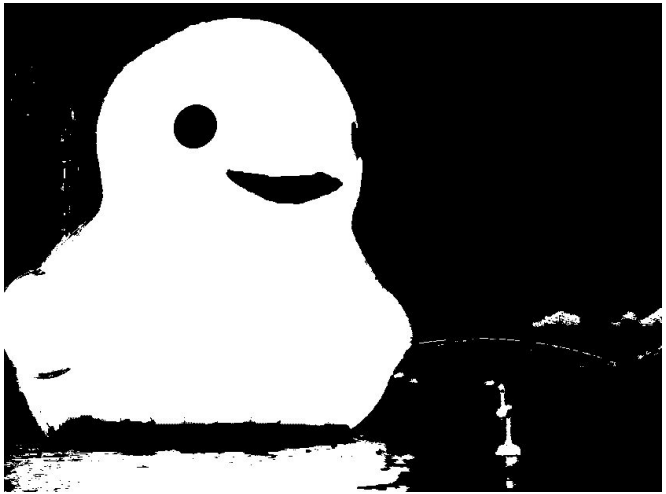
Operaciones morfológicas

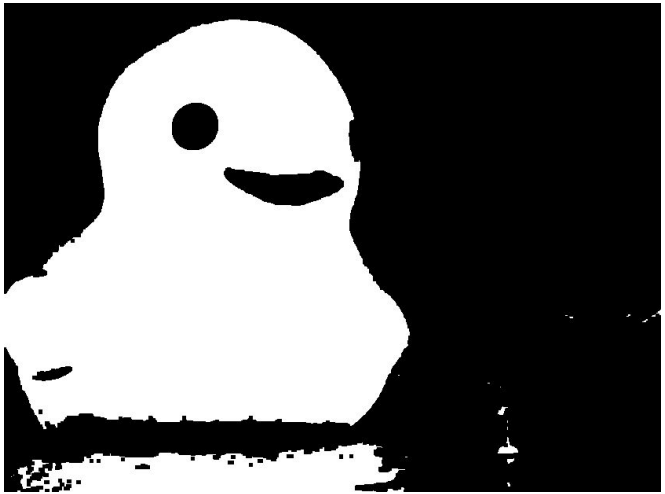
Erode



Operaciones morfológicas

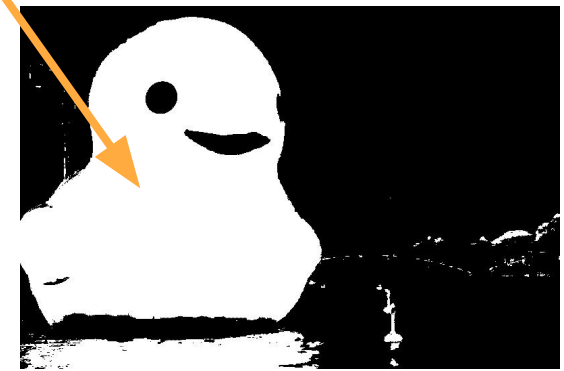
Erode

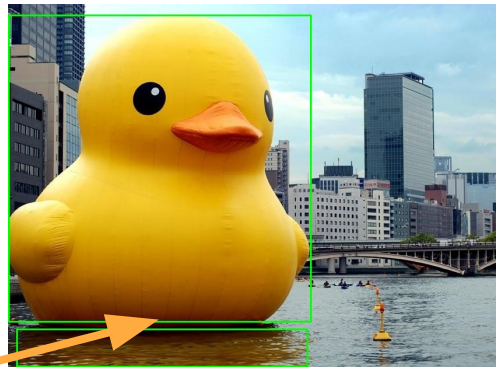




Detección de objetos

Blob (**B**inary **L**arge **OB**ject) : Grupo de píxeles conectados en una imagen binaria.





Bounding box: Polígono que encierra al blob

{

Tarea de hoy

Implementar un sistema de segmentación por color

}

ANEXO

Comandos útiles

- Cargar imagen

```
image = cv2.imread("carpeta/image.jpg")
```

- Mostrar imagen en OpenCV

```
cv2.imshow('nombre', image_bgr)
```

- Filtrar píxeles entre 2 límites (útil para escala de grises)

```
mask = cv2.inRange(image, lower_limit, upper_limit)
```

Comandos útiles

- Cambiar entre espacios de colores

```
image_out = cv2.cvtColor(image, color_space)
```

color_space:

cv2.COLOR_RGB2HSV

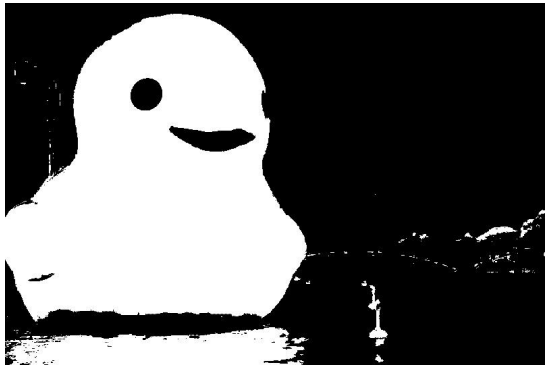
cv2.COLOR_RGB2BGR

cv2.COLOR_BGR2GRAY

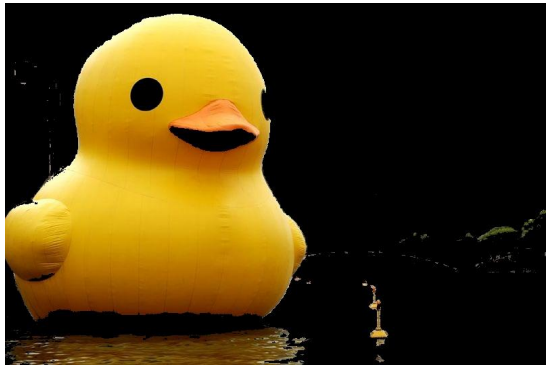
Comandos útiles

```
image_out = cv2.bitwise_and(image, image, mask= mask)
```

image



mask



image_out

Comandos útiles

- Transformación morfológicas

```
kernel = np.ones((5,5),np.uint8)
img_out = cv2.erode(img, kernel, iterations = 1)
img_out = cv2.dilate(img, kernel, iterations = 1)
```

- Buscar contornos de blobs

```
contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

Comandos útiles

- Obtener rectángulo

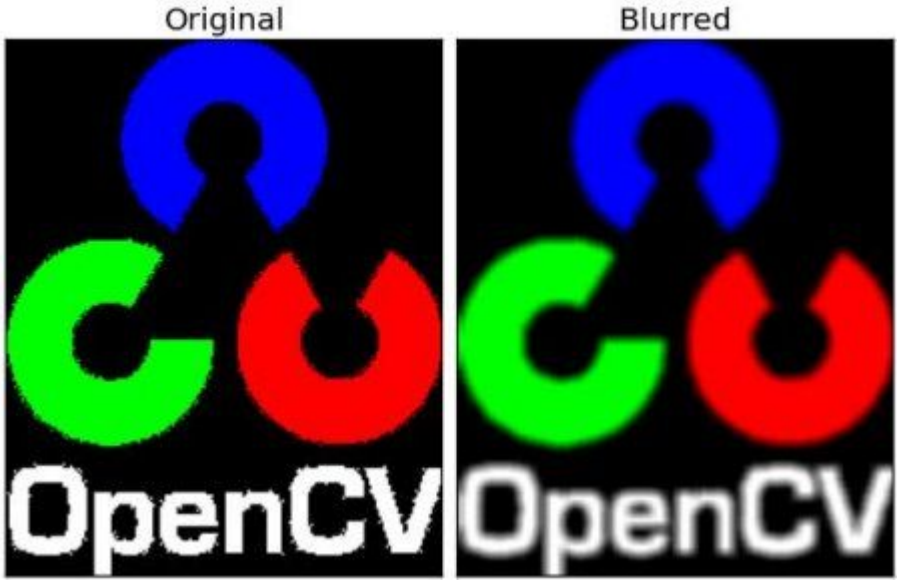
```
x,y,w,h = cv2.boundingRect(cnt)
```

- Dibujar rectángulo en imagen

```
cv2.rectangle(img, (x1,y1), (x2,y2), (0,0,0), 2)
```

Limpieza y filtrado de imágenes

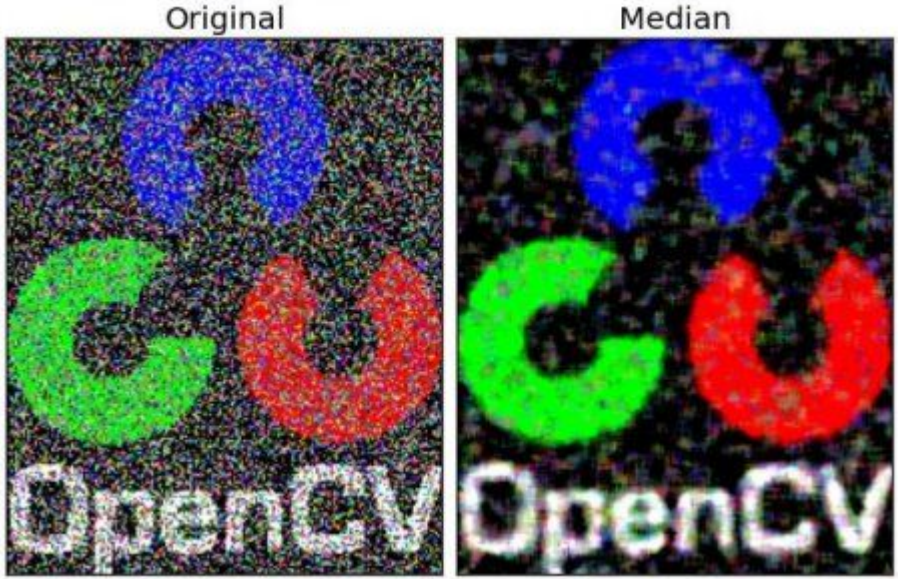
- **Gaussian blur:** In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, cv.GaussianBlur(). We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.



blur = cv.GaussianBlur(img,(5,5),0)

Limpieza y filtrado de imágenes

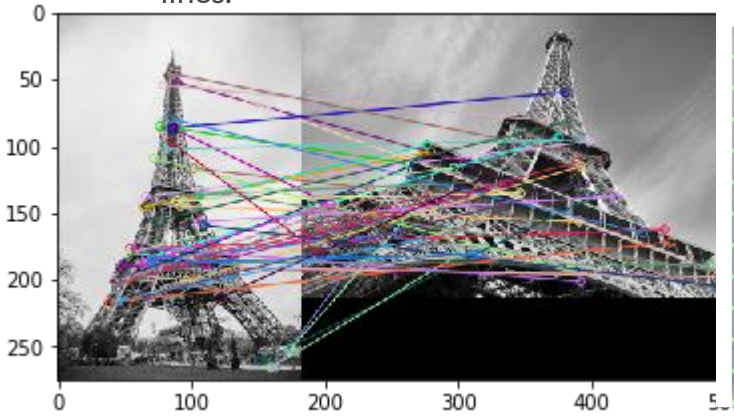
- **Median blur**: Here, the function cv.medianBlur() takes the median of all the pixels under the kernel area and the central element is replaced with this median value. It reduces the noise effectively. Its kernel size should be a positive odd integer.



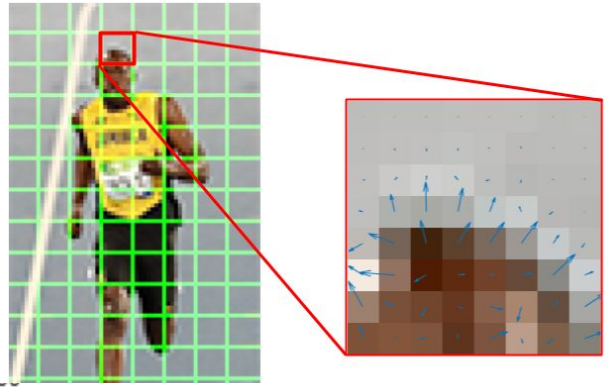
median = cv.medianBlur(img,5)

Extracción y selección de características

- **SIFT:** Obtención de puntos de interés, match de descriptores.
- **HOG:** Obtención de descriptores a partir del gradiente
- **HOUGH:** The Hough Line Transform is a transform used to detect straight lines.



Demo SIFT

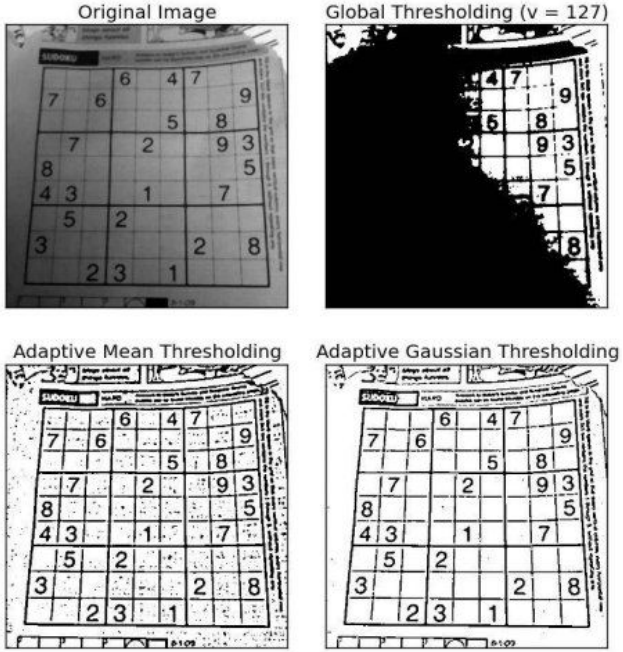


HOG

HOUGH lines

Extracción y selección de características

- Thresholding, clustering-based methods (K-Means).

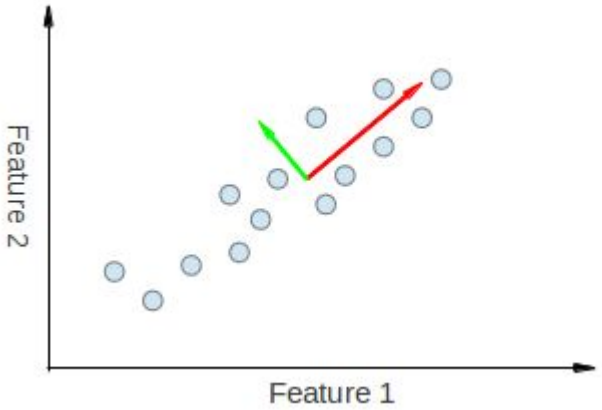


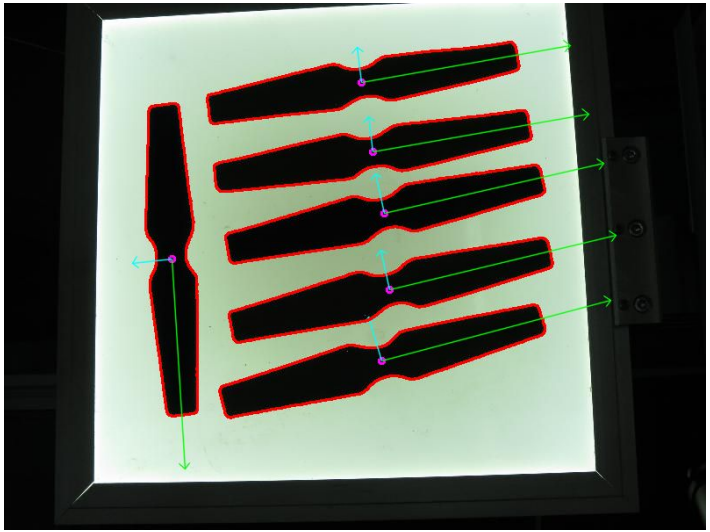
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html

Reducción de dimensionalidad

- Principal Component Analysis (PCA) : Principal Component Analysis (PCA) es un procedimiento estadístico que extrae las características más relevantes de un dataset. Útil para obtener orientación de objetos.





Tarea

{ Nos vamos al fablab primero :) }