Visión por computadora

Robótica y automatización

Facultad de Ingeniería - Instituto de Computación

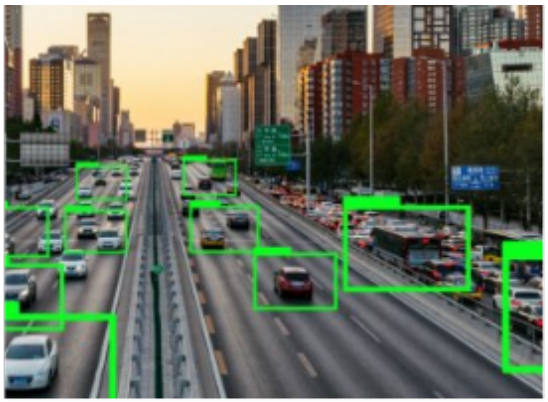
Visión por computadora

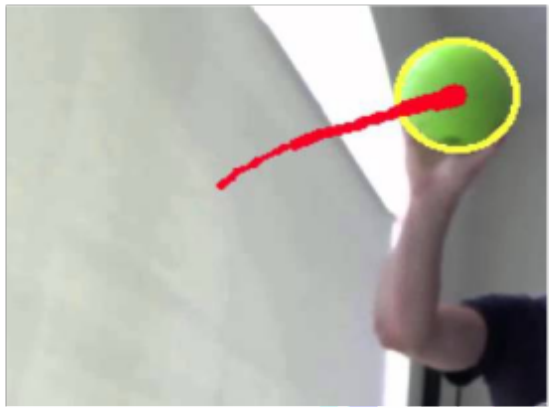
Conjunto de algoritmos que permiten obtener una representación visual del mundo, suficiente para la realización de una tarea dada

Visión por computadora

Tareas relacionadas con la robótica:

- Reconocer determinado objeto
- Determinar la pose de un objeto
- Seguir un objeto
- Evitar un obstáculo
- Ubicarse en el espacio, etc..

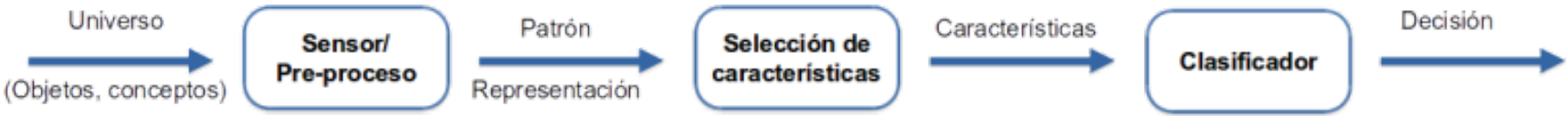




Visión por computadora

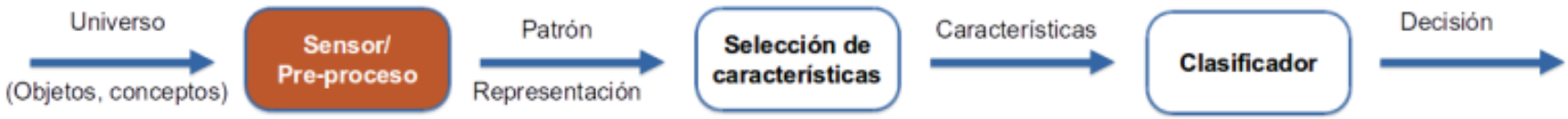
Reconocer un objeto

El reconocimiento de patrones es el estudio de cómo las máquinas observando el entorno pueden aprender a distinguir patrones de interés de un fondo, y tomar decisiones acertadas y razonables acerca de la categoría de los mismos (Anail Jain).



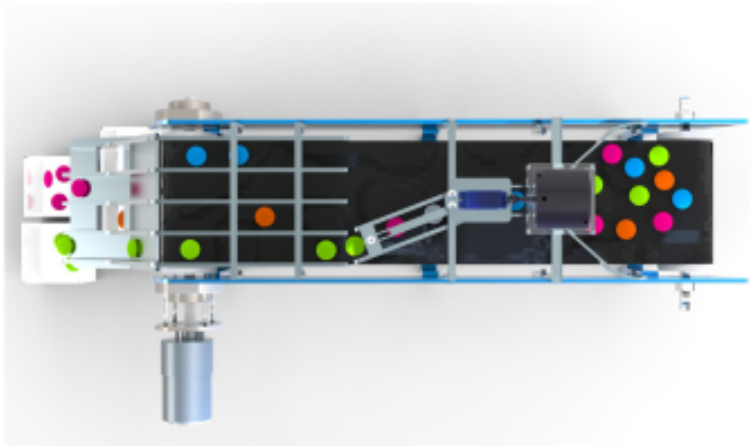
Visión por computadora

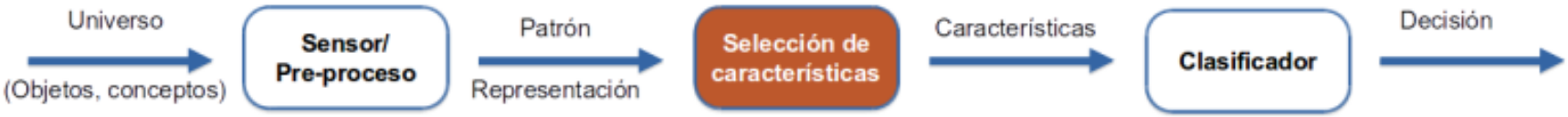
Esquema clásico del reconocimiento de patrones



Visión por computadora

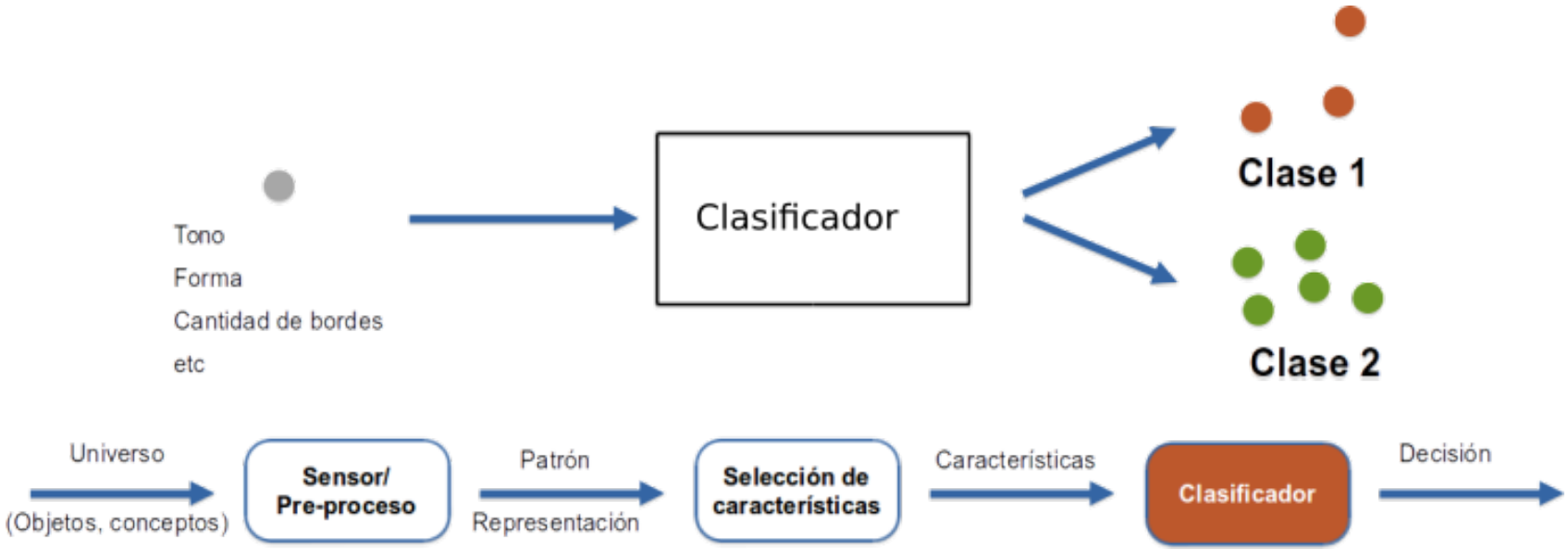
Esquema clásico del reconocimiento de patrones





Visión por computadora

Esquema clásico del reconocimiento de patrones

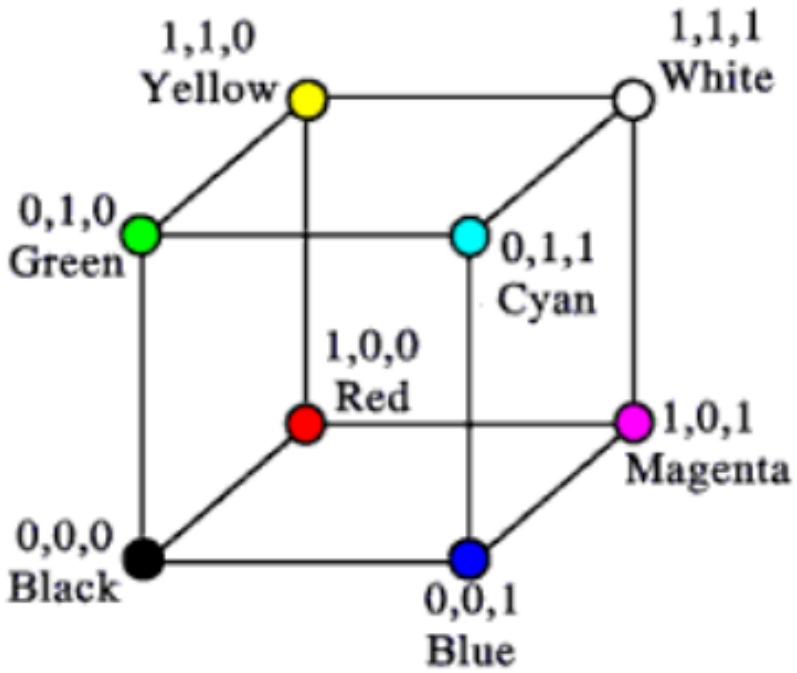


El color

- Diferentes formas de representarlo

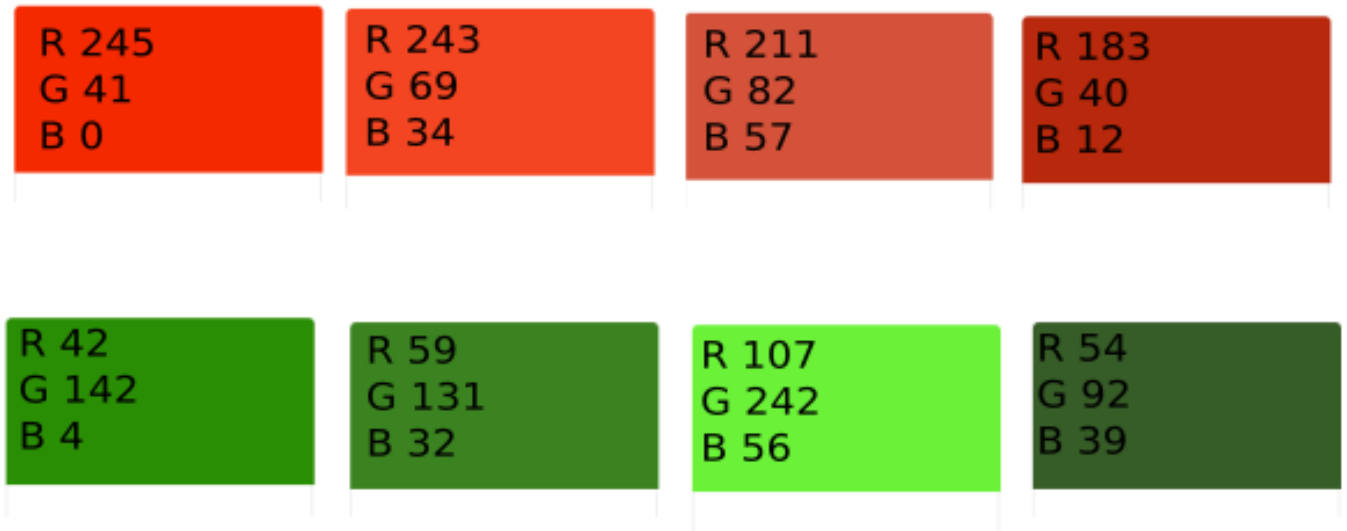
El color

- Diferentes formas de representarlo
- RGB



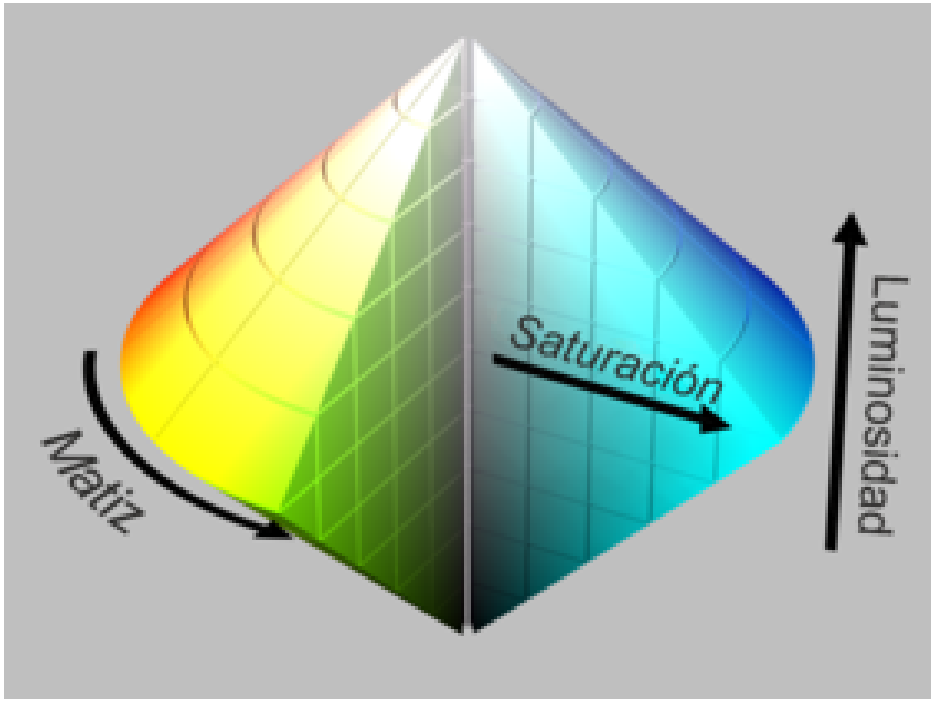
El color

- Diferentes formas de representarlo
- RGB



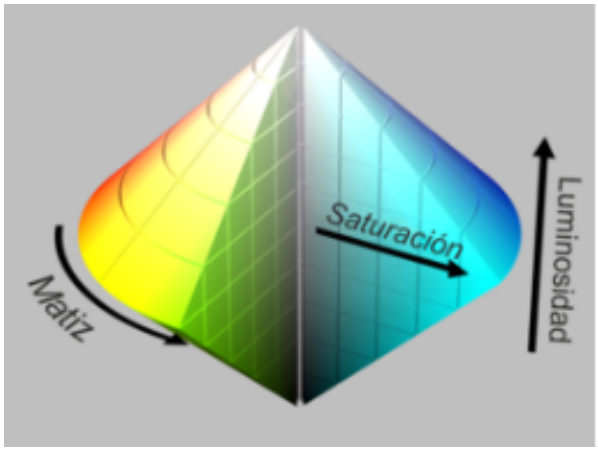
El color

- Diferentes formas de representarlo
- HSL



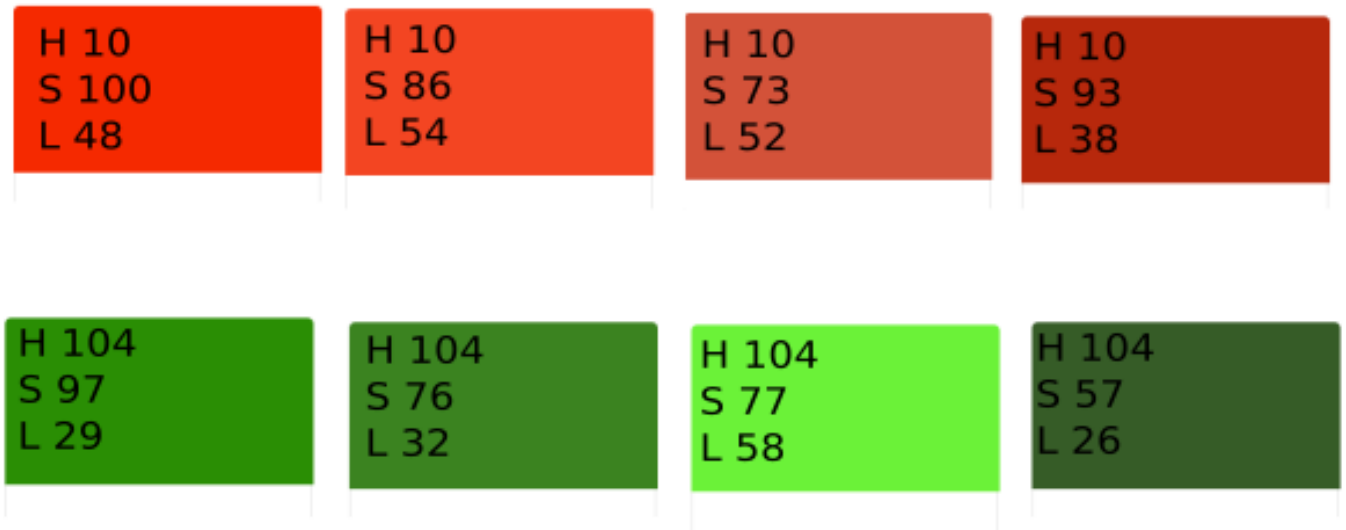
El color

- Diferentes formas de representarlo
- HSL



El color

- Diferentes formas de representarlo
- HSL



Utilizando OpenCV

- Bajar el programa color.py del eva del curso
- Conectar la cámara y correr el programa

Utilizando OpenCV

- Convertir la imagen a HSV

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(True):
    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Utilizando OpenCV

- Filtrar por color

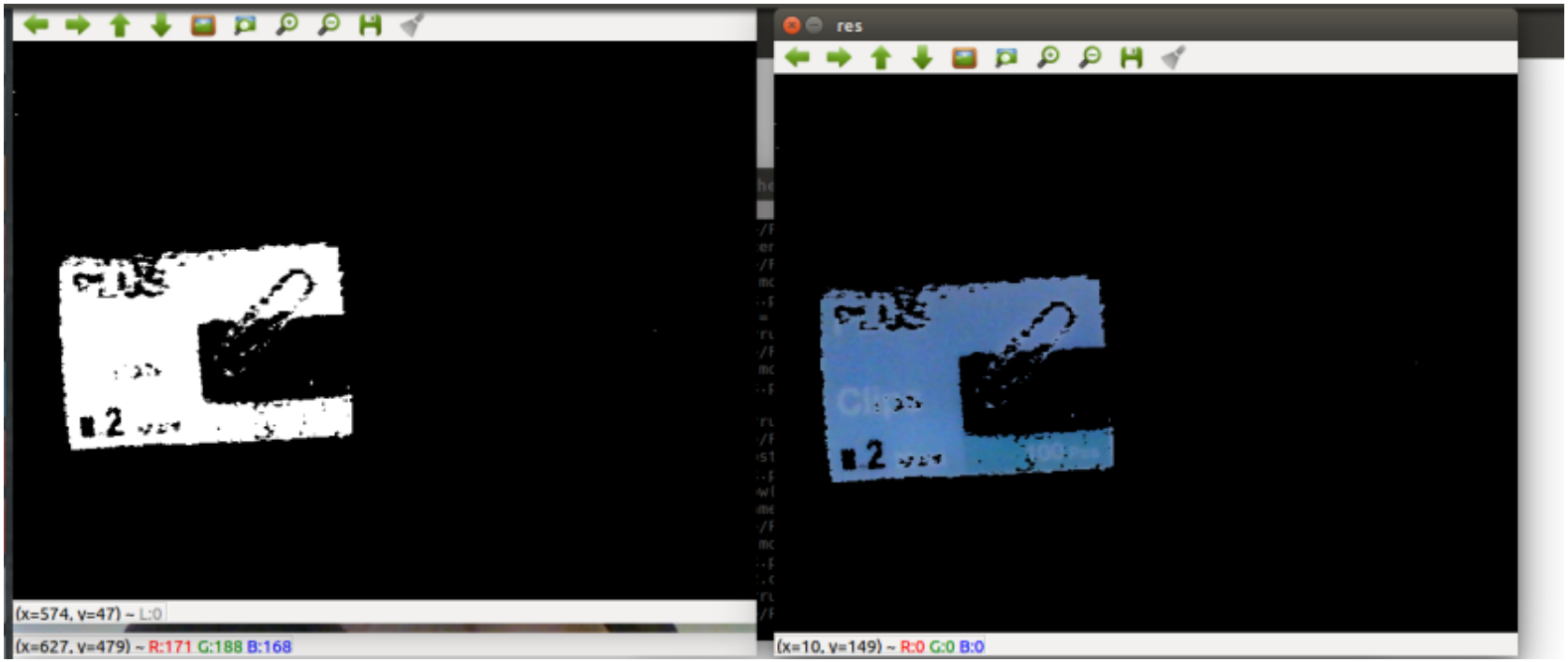
Utilizando OpenCV

- Filtrar por color

```
# Convert BGR to HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# define color range  in HSV
lower_range = np.array([0,100,100])
upper_range = np.array([5,255,255])

# Threshold the HSV image to get only in range colors
mask = cv2.inRange(hsv, lower_range, upper_range)

cv2.imshow('frame',frame)
cv2.imshow('mask',mask)

k = cv2.waitKey(5) & 0xFF
if k == 27:
    break

cv2.destroyAllWindows()
```

Utilizando OpenCV

- Realizar mascara para ver la imagen en color



Utilizando OpenCV

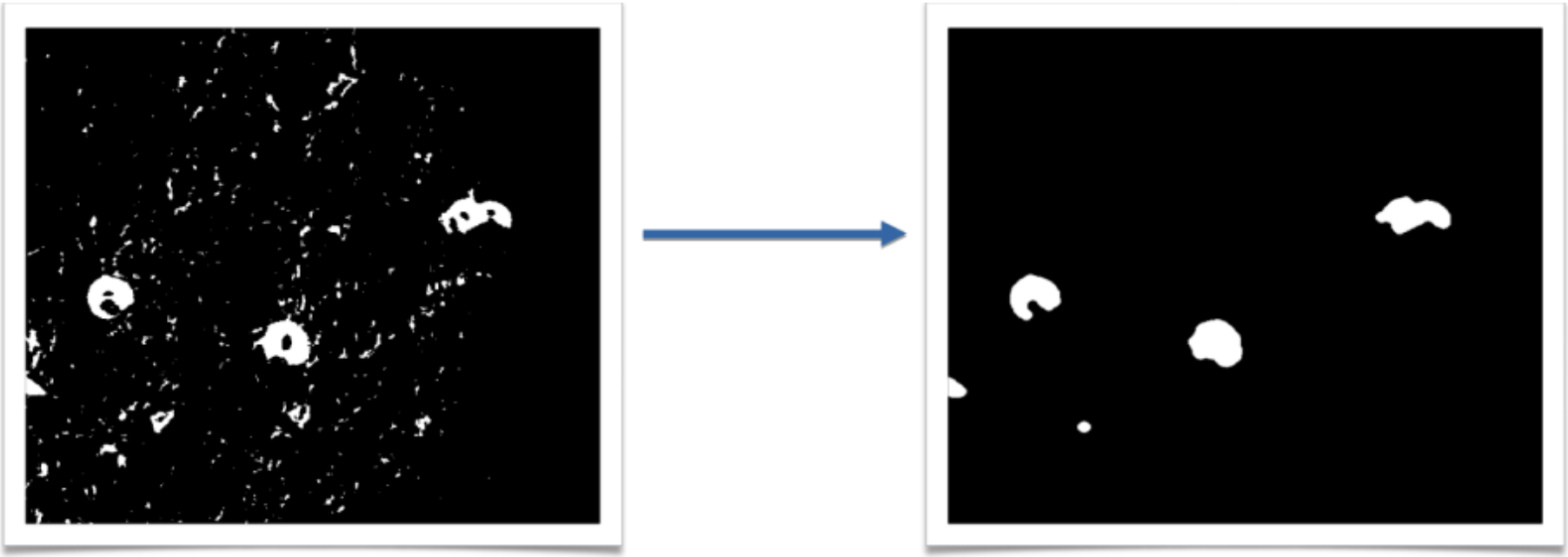
- Realizar mascara para ver la imagen en color

```
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv, lower_range, upper_range)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask= mask)
```

Utilizando OpenCV

- Modificar el programa color.py para que muestre la mascara en color
- Ajustar los valores del color para reconocer un objeto del color que ustedes elijan

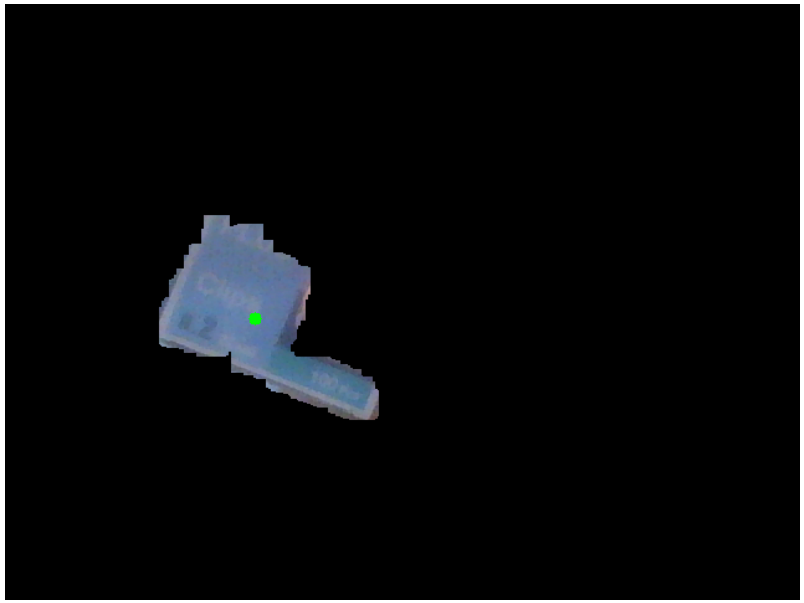
- Operaciones morfológicas



- Operaciones morfológicas

```
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv, lower_range, upper_range)

kernel =cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
mask = cv2.erode(mask, kernel, iterations=2)
mask = cv2.dilate(mask, kernel, iterations=4)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask= mask)
```

- Centro del objeto



- Centro del objeto

```
#Detectamos los contornos
_,contours,_ = cv2.findContours(mask, cv2.RETR_TREE,
                                cv2.CHAIN_APPROX_SIMPLE)
if contours != []:
    # Si hay contorno nos quedamos con el mayor
    mayor_contorno = max(contours, key = cv2.contourArea)
    momentos = cv2.moments(mayor_contorno)
    # Dibujamos el centro
    cx = float(momentos['m10']/momentos['m00'])
    cy = float(momentos['m01']/momentos['m00'])
    cv2.circle(res, (int(cx), int(cy)), 5, (0,255,0), -1)
```

Utilizando OpenCV

- Modificar el programa anterior para mostrar el centro del objeto de mayor área

Preguntas