Visión Computacional Aplicada a la Robótica

Dr. Marco Negrete

Facultad de Ingeniería, UNAM

Semestre 2023-2

Presentación del curso

Objetivos:

- Aprender los conceptos básicos de visión computacional enfocados a la solución de un problema particular: dotar de autonomía a un robot móvil
- Implementar dichos conceptos en un ambiente simulado
- Familiarizar al estudiante con diversas herramientas de software como ROS, Numpy y OpenCV

Contenido

- Introduction
- Procesamiento de Imágenes
- Modelo de una cámara
- Espacios de color
- Características (*Features*)
- Clasificación y reconocimiento
- Redes Neuronales
- Navegación visual

Bibliografía recomendada:

- `https://drive.google.com/drive/folders/1gb7VQJG5eUkCvCginRHHGn5lez6VASBJ?usp=sharing`
- `https://drive.google.com/drive/folders/1Epl2b51xEJzCvzfugBD1i7xGdKYdJucy?usp=sharing`

Forma de trabajo

- Horario: Viernes de 10:00 a 13:00.
- Para la realización de ciertas tareas se utilizará el simulador Gazebo junto con la plataforma ROS Noetic. Este software corre en el sistema operativo Ubuntu 20.04. Las prácticas implicarán escribir código para lo cual se utilizará el repositorio
  `https://github.com/mnegretev/ComputerVisionForRobotics-2023-2`
  Si no se desea instalar el sistema operativo de forma nativa, en el repositorio hay una máquina virtual con todo el software ya instalado.
- Para el uso del material del curso, se recomienda manejar las siguientes herramientas:
  - Sistema operativo Ubuntu
  - Lenguajes Python y C++
  - Software de control de versiones Git

  Un conocimiento a nivel introductorio es suficiente.
- Los códigos de cada práctica se subirán al repositorio en la rama asignada para cada estudiante.
- Los reportes escritos se entregarán al inicio de la clase en la fecha asignada.

Habrá un grupo en Google Classroom para todos los avisos.

Forma de evaluar

Rubros a evaluar:

Prácticas	50 %
Examen	50 %

¿Qué es la visión computacional?

- **Visión Humana:** Se puede concebir como una tarea de procesamiento de información, que obtiene significado a partir de los estímulos percibidos por los ojos.
- **Visión Computacional:** Desarrollo de programas de computadora que puedan *interpretar* imágenes. Es decir, realizar la visión humana por medios computacionales.

Teoría de Marr

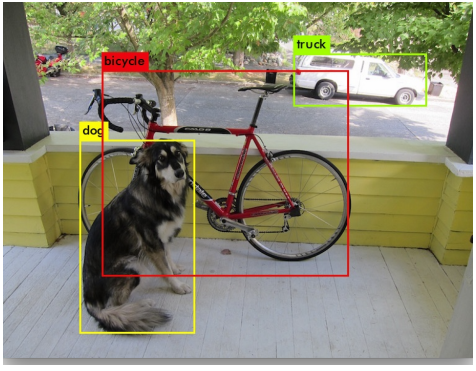
"Visión es un proceso que produce, a partir de imágenes del mundo externo, una descripción que es útil para el observador y que está libre de información irrelevante." (Marr, 1976).
El fenómeno de la visión lo podemos considerar como el producto de un sistema de procesamiento de información.
Marr propone los siguientes tres niveles de construcción de un sistema de procesamiento de información:

1. Teoría Computacional (¿Cuál es el problema por resolver?)
2. Representación y algoritmos (Estrategia usada para resolverlo)
3. Implementación (Realización física, software y hardware)

Es decir, la visión computacional sería un proceso parecido a la visión humana, similar en los niveles computacionales y de algoritmos, pero implementado de forma diferente: en hardware de procesamiento con sensores de visión.

Visión Computacional

Por lo tanto, la tarea de la Visión por computadora es la construcción de descriptores de la escena con base en características relevantes contenidas en una imagen:



- Objetos
- Formas de Superficies
- Colores
- Texturas
- Movimientos
- Iluminación
- Reflejos

Vision Computacional vs Proc de Imágenes

1. Procesamiento de imagenes: Es cualquier forma de procesamiento de señales donde la entrada es una imagen, la salida puede ser otra imagen o un conjunto de características o parámetros relacionados con la misma.
2. Visión Computacional: Estudio y aplicación de métodos que permiten a las computadoras "entender" el contenido de una imagen.
3. Visión Máquina: Es la aplicación de la visión por computadora en la industria y procesos de manufactura.

Aplicaciones

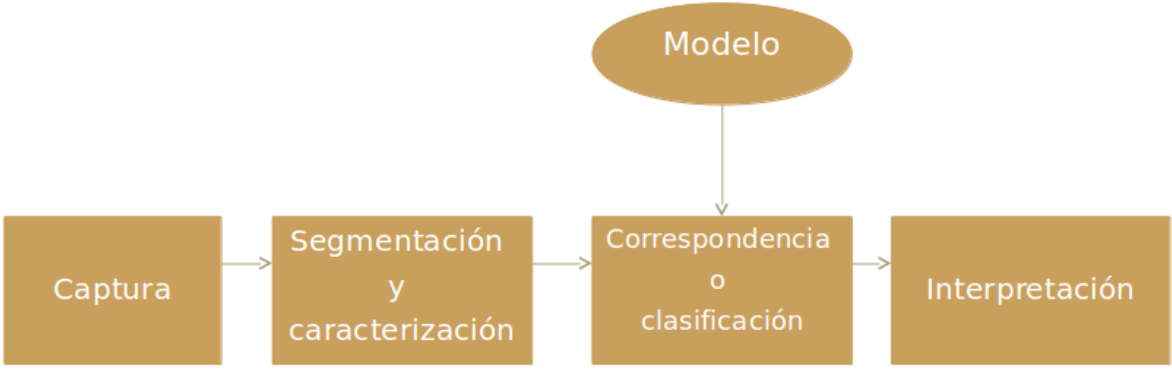
Tareas que se pueden hacer con visión computacional (con aplicaciones a la robótica):

- OCR (Optical Character Recognition)
- Detección e identificación de rostros
- Reconocimiento de objetos
- Percepción para vehículos sin conductor
- Reconocimiento de gestos

Otras aplicaciones:

- Vigilancia
- Imagenología médica
- Consultas a bases de datos de imágenes.
- Percepción remota

Esquema de Visión



Dificultades

El entorno real tiene una gran cantidad de variaciones en las imágenes de entrada.

1. Iluminación
2. Orientación
3. Oclusión
4. Escala
5. Ruido
6. Desenfoque

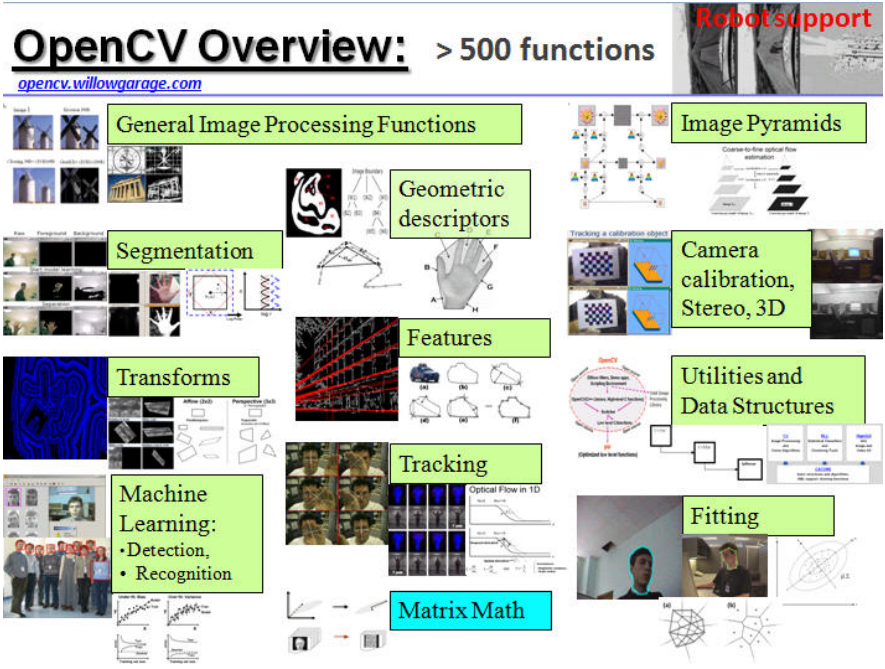
Hardware y Software

La computación con imágenes tiene mas de 30 años, sin embargo, en los últimos años, se ha incrementado considerablemente su desarrollo debido a:

1. Decremento en los precios
2. Memoria con gran capacidad
3. Procesadores de propósito general de alta velocidad.
4. Existen scanners o camaras digitales que pueden ser utilizados para procesar imágenes propias.
5. Existen bibliotecas de software que contienen subrutinas de procesamiento de imágenes (opencv).

OpenCV

OpenCV es una biblioteca libre de visión artificial, originalmente desarrollada por Intel.
Programación en código Python, C y C++
Existen versiones para GNU/Linux, Mac OS X y Windows


OpenCV Overview:
> 500 functions
opencv.willowgarage.com
Robot support
General Image Processing Functions
Image Pyramids
Geometric descriptors
Segmentation
Camera calibration, Stereo, 3D
Features
Transforms
Utilities and Data Structures
Tracking
Machine Learning:
•Detection,
• Recognition
Optical Flow in 1D
Fitting
Matrix Math

Funciones básicas

En Python, OpenCV está basado en la biblioteca Numpy por lo que muchas operaciones con imágenes corresponden a operaciones de Numpy sobre matrices. Algunas funciones que se usarán frecuentemente a lo largo del curso son:

- `numpy.zeros`
- `enumpy.ones`
- `cv2.imread`
- `cv2.bitwise_and`
- `cv2.bitwise_or`
- `cv2.bitwise_not`
- `cv2.bitwise_xor`
- `cv2.inRange`
- `cv2.merge`
- `cv2.split`
- `cv2.imshow`
- `cv2.VideoCapture`
- `cv2.mean`

Ejercicio 1 - Lectura de imágenes y manejo de matrices

Escriba el siguiente código en un editor de texto:

```python
import cv2
import numpy as np

def main():
    img_baboon = cv2.imread('baboon.jpg')
    img_blank  = np.zeros((512, 512, 3), np.uint8)
    img_blank[128:256,0:256] = 255*np.ones((128, 256, 3), np.uint8)
    img_roi    = img_baboon[120:360, 160:480]
    cv2.imshow("BGR Image", img_baboon)
    cv2.imshow("Blank image", img_blank)
    cv2.imshow("Region of Interest", img_roi)
    cv2.waitKey(0)

if __name__ == '__main__':
    main()
```

Descargue la imagen `baboon.jpg` en la misma carpeta que el código. Abra una terminal y ejecute el código con el comando:

```
python3 Exercise01.py
```

Modifique manualmente el tamaño de la imagen y modifique el código para que funcione con la imagen modificada.

Operaciones lógicas

- Las operaciones lógicas se usan generalmente para implementar *máscaras*
- Puesto que son operaciones bit a bit, las imágenes a operar deben ser del mimo tamaño y tipo (mismo número de canales y mismo tipo de dato)
- Las máscaras suelen provenir de operaciones que dan como resultado imágenes binarias, como operaciones de comparación, umbralización o detección de bordes.

Ejercicio 2 - Operaciones lógicas bit a bit

Escriba el siguiente código en un editor de texto:

```
import cv2
import numpy

def main():
    img_blank  = numpy.zeros((512, 512, 3), numpy.uint8)
    img_blank[128:256,0:256] = 255*numpy.ones((128, 256, 3), numpy.uint8)
    img_baboon = cv2.imread('baboon.jpg')
    img_and    = cv2.bitwise_and(img_baboon, img_blank)
    img_or     = cv2.bitwise_or (img_baboon, img_blank)
    img_not    = cv2.bitwise_not(img_baboon)
    img_xor    = cv2.bitwise_xor(img_baboon, img_blank)
    cv2.imshow("OR Operator", img_or)
    cv2.imshow("XOR Operator", img_xor)
    cv2.imshow("AND Operator", img_and)
    cv2.imshow("NOT Operator", img_not)
    cv2.waitKey(0)

if __name__ == '__main__':
    main()
```

Ejecute el código y observe lo que hace cada operación.

Ejercicio 3 - Controles gráficos

- OpenCV provee controles de interfaz gráfica de usuario que son útiles para la depuración de programas.
- Las barras de seguimiento (*trackbars*) sirven para fijar valores en un intervalo y disparan una función *callback* cuando hay un cambio en el valor.
- También provee *callbacks* para el manejo de eventos del mouse (clicks derecho e izquierdo, movimiento del puntero, etc.)

Descargue el archivo `Exercise03.py`, ejecútelo y observe el comportamiento de la barra de seguimiento y de los clicks derecho e izquierdo. Consulte la documentación en línea y modifique el código para que se dibujen rectángulos con dimensiones dadas por dos clicks.

Manejo de video

- OpenCV provee la función `VideoCapture` para abrir fácilmente una cámara de video
- El video se entrega como una secuencia de imágenes por lo que un programa que maneje video debe contener un cilo *while*
- La resolución más común es la de VGA de 640x480 aunque se pueden usar otras resoluciones dependiendo de las capacidades de cada cámara
- El número de cuadros por segundo (*fps*) suele ser de 30, aunque puede ser de 60 o más, dependiendo de las capacidades de la cámara
- Si hay más de una cámara conectada, OpenCV las enumera con índices $i = 0, 1, 2, 3, \ldots$ y se puede elegir qué cámara abrir usando dicho índice
- La misma funcion `VideoCapture` se puede usar para abrir videos en distintos formatos

Ejercicio 4 - Manejo de vídeo

Escriba el siguiente código en un editor de texto:

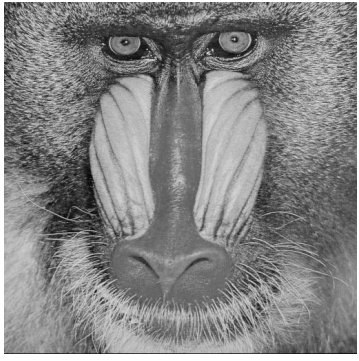
```
import numpy as np
import cv2

cap  = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    cv2.imshow('My Video', frame)
    if cv2.waitKey(10) & 0xFF == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

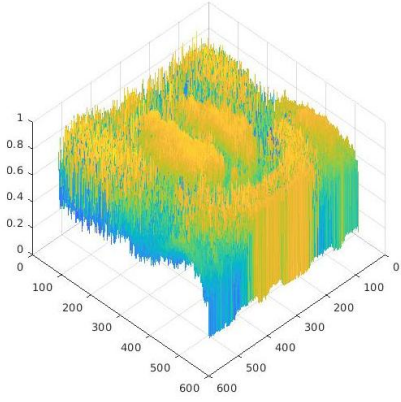
El código anterior sólo abre una cámara y despliega la imagen en una ventana. Modifique el código para que realice lo siguiente:

- Cree una imagen en blanco y negro de dimensiones apropiadas similar a la del ejercicio 2
- Realice una o más operaciones lógicas entre el video y la imagen en blanco y negro
- Despliegue el resultado en una ventana

Imágenes como funciones

- Una imagen (en escala de grises) es una función $I(x, y)$ donde x, y son variables discretas en coordenadas de imagen y la función I es intensidad luminosa.
- Las imágenes también pueden considerarse como arreglos bidimensionales de números entre un mínimo y un máximo (usualmente 0-255).
- Aunque formalmente una imagen es un mapeo $f : \mathbb{R}^2 \to \mathbb{R}$, en la práctica, tanto x, y como I son varialbes discretas con valores entre un mínimo y un máximo.
- Las imágenes de color son funciones vectoriales $f : \mathbb{R}^2 \to \mathbb{R}^3$ donde cada componente de la función se llama canal.





Operaciones básicas

Las operaciones básicas con una imagen son las mismas que con una señal cualquiera:

- Desfase
- Escalamiento
- Inversión en x, y
- Suma y Resta
- Multiplicación

Ver ejercicios en Python.

Tipos de ruido

El ruido es una señal aleatoria $\eta(x, y)$, es decir, no sabemos cuánto vale para un punto determinado (x, y) pero sí podemos caracterizarla.

$$I_n(x, y) = I(x, y) + \eta(x, y)$$

Existen varios tipos de ruido:

- Sal y pimienta: aleatoriamente aparecen puntos ya sea blancos o negros
- Ruido de impulso: aleatoriamente aparecen puntos blancos
- Ruido gausiano: $\eta(x, y)$ se distribuye

Ver ejercicios en Python

SLID

Un sistema S es un mapeo del conjunto de señales al conjunto de señales, es decir, es algo donde entra una señal $I(x, y)$ y sale otra señal $O(x, y)$:

$$O(x, y) = S[I(x, y)]$$

Los sistemas lineales invariantes ante el desfase son sistemas en los que se cumplen las siguientes propiedades:

- **Aditividad y homogeneidad:**

$$S[\alpha I_1(x, y) + \beta I_2(x, y)] = \alpha S[I_1(x, y)] + \beta S[I_2(x, y)]$$

- **Invarianza ante el desfase:**

$$\text{Si} \qquad S[I(x, y)] = O(x, y) \qquad \text{entonces:}$$

$$S[I(x - i, y - j)] = O(x - i, y - j) \qquad \forall i, j \in \mathbb{Z}$$

Los SLID se pueden caracterizar de varias formas:

- Ecuaciones en diferencias
- Funciones de transferencia
- Respuesta al impulso

Convolución

Si se conoce la respuesta al impulso $H(x, y)$ de un sistema SLID, se puede obtener la salida $O(x, y)$ ante cualquier entrada $I(x, y)$, mediante la convolución, definida como:

$$O(x,y) = I(x,y) * H(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j)H(x-i, y-j)$$

Ejemplos:

$$\begin{bmatrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 2 & 6 \\ 5 & 3 & 5 & 8 \\ 9 & 7 & 9 & 3 \end{bmatrix} * [1 \quad -1] = \begin{bmatrix} 3 & -2 & 3 & -3 & -1 \\ 5 & 4 & -7 & 4 & -6 \\ 5 & -2 & 2 & 3 & -8 \\ 9 & -2 & 2 & -6 & -3 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 2 & 6 \\ 5 & 3 & 5 & 8 \\ 9 & 7 & 9 & 3 \end{bmatrix} * \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 4 & 1 \\ 2 & 8 & -2 & 5 \\ 0 & -6 & 3 & 2 \\ 4 & 4 & 4 & -5 \\ -9 & -7 & -9 & -3 \end{bmatrix}$$

Manejo de bordes

En el ejemplo anterior, supusimos que fuera de la matriz, todos los elementos son cero. Sin embargo existen otras formas de manejar los borde:

- Recortar: suponer que fuera de la matriz los valores son cero. En el caso de una imagen, suponemos pixeles negros fuera de la imagen.
- Wrap around: suponer que la imagen es periódica.
- Borde repetido: suponer que los valores de los bordes se mantienen iguales fuera de la imagen.
- Reflexión: fuera de los bordes se tiene una imagen en espejo.

Propiedades de la convolución

- Es conmutativa: $H * I = I * H$
- Es asociativa: $H * I_1 * I_2 = H * (I_1 * I_2) = (H * I_1) * I_2$
- Es distributiva: $H * (I_1 + I_2) = H * I_1 + H * I_2$
- Es lineal: $H * (\alpha I_1 + \beta I_2) = \alpha H * I_1 + \beta H * I_2$

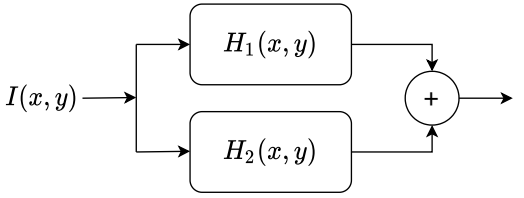
En el caso de secuencias finitas bidimensionales:

- Si $I \in \mathbb{R}^{r_1 \times c_1}$ y $H \in \mathbb{R}^{r_2 \times c_2}$, entonces $(I * H) \in \mathbb{R}^{(r_1+r_2-1)\times(c_1+c_2-1)}$
- Si $I \in \mathbb{R}^{r_1 \times c_1}$ y $H \in \mathbb{R}^{r_2 \times c_2}$, la complejidad de la convolución es del orden de $r_1 r_2 c_1 c_2$

Conexión de sistemas SLID

Dos o más SLID se pueden conectar de dos formas distintas:

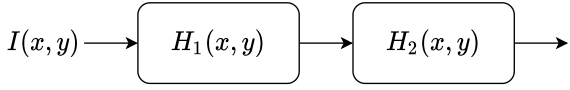
- Conexión en paralelo:



$$O(x,y) = I(x,y) * [H_1(x,y) + H_2(x,y)]$$

$$H(x,y) = H_1(x,y) + H_2(x,y)$$

- Conexión en cascada:



$$O(x,y) = I(x,y) * [H_1(x,y) * H_2(x,y)]$$

$$H(x,y) = H_1(x,y) * H_2(x,y)$$

Gradiente

El gradiente de una imagen está definido como:

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

Las derivadas parciales se puede aproximar mediante diferencias finitas:

$$\begin{aligned} \frac{\partial I}{\partial x} &= \lim_{\Delta x \to 0} \frac{I(x + \Delta x, y) - I(x, y)}{\Delta x} \approx I_{i,j} - I_{i,j-1} \\ \frac{\partial I}{\partial y} &= \lim_{\Delta y \to 0} \frac{I(x, y + \Delta y) - I(x, y)}{\Delta y} \approx I_{i,j} - I_{i-i,j} \end{aligned}$$

donde (i, j) representan las coordenadas de imagen renglón-columna. Estas diferencias finitas se puede obtener mediante una convolución:

$$\begin{aligned} \frac{\partial I}{\partial x} &\approx I * \begin{bmatrix} 1 & -1 \end{bmatrix} \\ \frac{\partial I}{\partial y} &\approx I * \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{aligned}$$

Gradiente

Una mejor aproximación de la derivada es no solo tomar la diferencia entre el valor actual y el anterior $(I_{i,j} - I_{i-1,j})$, sino promediarlo con la diferencia $(I_{i+1,j} - I_{i,j})$:

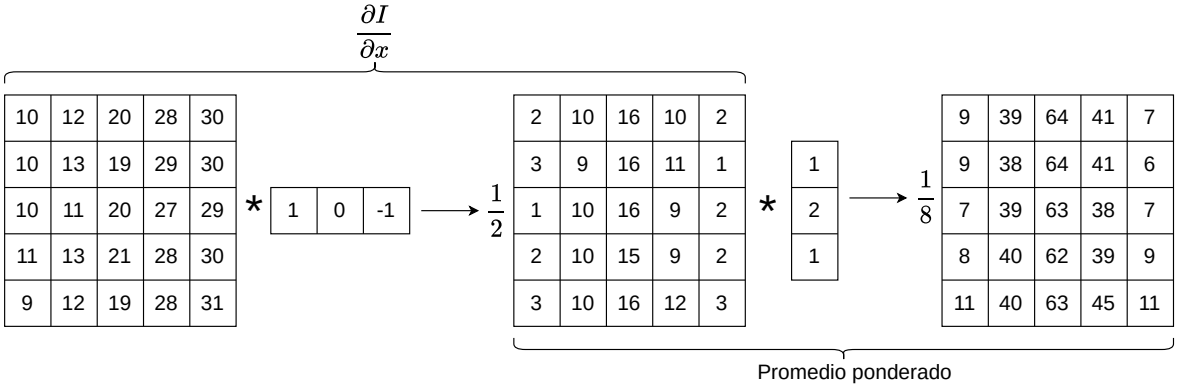
$$\frac{1}{2}[(I_{i,j} - I_{i-1,j}) + (I_{i+1,j} - I_{i,j})] = \frac{1}{2}(I_{i+1,j} - I_{i-1,j})$$

Generalmente se ignora el coeficiente y se utilizan los siguientes Kernels:

$$\begin{aligned} \frac{\partial I}{\partial x} &\approx I * [1 \quad 0 \quad -1] \\ \frac{\partial I}{\partial y} &\approx I * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \end{aligned}$$

El filtro de Sobel

El Operador de Sobel o Filtro de Sobel consiste en un Kernel que permite obtener las derivadas parciales, aproximadas por diferencias finitas, y promediadas con un filtro Gaussiano:



Se realiza un proceso similar para la derivada parcial en Y. Aplicando la propiedad asociativa de la convolución, se obtienen los siguientes ekernels:

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Ejemplo

Magnitud y Ángulo

El gradiente en cada pixel de la imagen se puede calcular mediante la approximación de las derivadas parciales:

$$\begin{aligned} \frac{\partial I}{\partial x} &\approx I * Sx = G_x \\ \frac{\partial I}{\partial y} &\approx I * Sy = G_y \end{aligned}$$

En la mayoría de las aplicaciones es más últil expresar el gradiente en forma polar:

$$\nabla I = G_m \angle G_a$$

Donde la magnitud del gradiente y la fase, para cada pixel, se calculan como:

$$\begin{aligned} G_{m_{i,j}} &= \sqrt{G_{x_{i,j}}^2 + G_{y_{i,j}}^2} \\ G_{a_{i,j}} &= \text{atan2}(G_{y_{i,j}}, G_{y_{i,j}}) \end{aligned}$$

Detector de Bordes de Canny

El detector de bordes de Canny es un detector basado en gradiente que consta de los siguientes pasos básicos:

1. Obtención del gradiente en magnitud y ángulo, mediante operadores de Sobel
2. Supresión de puntos no máximos
3. Aplicación de un doble umbral

Aunque no es un paso en sí del Detector de Canny, generalmente se considera como primer paso la aplicación de un filtro Gaussiano para disminuir el ruido.

Obtención del gradiente

Después del filtro Gaussiano, el primer paso es obtener el gradiente de la imagen mediante el Filtro de Sobel, en la forma de magnitud y ángulo:

10	12	20	28	30
10	13	19	29	30
10	11	20	27	29
11	13	21	28	30
9	12	19	28	31

$*$

1	0	-1
2	0	-2
1	0	-1

$=$

9	39	64	41	7
9	38	64	41	6
7	39	63	38	7
8	40	62	39	9
11	40	63	45	11

$x, y \rightarrow r\angle\theta$

9.05	39.01	64.00	41.01	7.07
9.05	38.05	64.03	41.10	7.21
7.61	39.11	63.07	38.00	7.07
8.24	40.00	62.00	39.11	11.40
13.03	40.44	63.19	45.01	11.40

10	12	20	28	30
10	13	19	29	30
10	11	20	27	29
11	13	21	28	30
9	12	19	28	31

$*$

1	2	1
0	0	0
-1	-2	-1

$=$

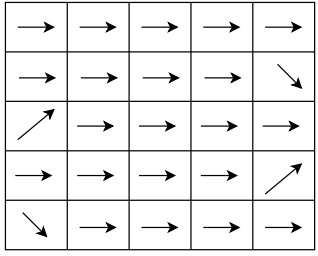
1	1	0	1	1
-1	-2	-2	-3	-4
3	3	3	0	-1
-2	0	0	3	7
-7	-6	-5	-1	3

0.11	0.02	0	0.02	0.14
-0.11	-0.05	-0.03	-0.07	-0.58
0.40	0.07	0.04	0	-0.14
-0.24	0	0	0.07	0.66
-0.56	-0.14	-0.07	-0.02	0.26

Supresión de no máximos

Este paso consiste en comparar la magnitud de cada pixel, con los pixeles anterior y posterior en la dirección del gradiente. Aunque la fase es un ángulo en $[-\pi, \pi]$, la dirección del gradiente se debe redondear a algún valor correspondiente a la connectividad 8: *N, NE, E, SE*. Debido a que el pixel se compara en la dirección positiva y negativa del gradiente, no es necesario considerar las direcciones *S, SW, W, NW*.

9.05	39.01	64.00	41.01	7.07
9.05	38.05	64.03	41.10	7.21
7.61	39.11	63.07	38.00	7.07
8.24	40.00	62.00	39.11	11.40
13.03	40.44	63.19	45.01	11.40



0	0	64.00	0	0
0	0	64.03	0	0
0	0	63.07	0	0
0	0	62.00	0	0
0	0	63.19	0	0

Para cada pixel p_i, considere p_{i+1}, el pixel siguiente en la dirección del gradiente, y p_{i-1}, el pixel anterior, en la dirección del gradiente. El valor para cada pixel q_i en la imagen resultante es:

$$q_i = \begin{cases} p_i & \text{si} \quad p_i > p_{i+1} \quad \text{y} \quad p_i > p_{i-1} \\ 0 & \text{en otro caso} \end{cases}$$

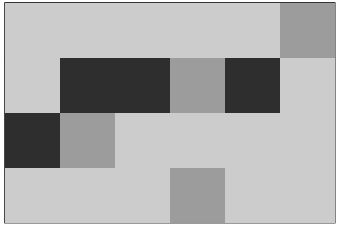
Aplicación de doble umbral

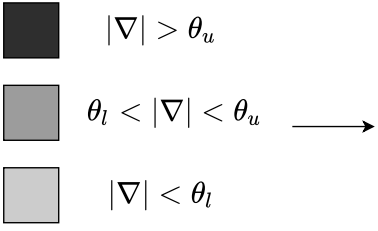
En este paso, se definen dos umbrales: superior θ_u e inferior θ_l. Los pixeles se clasifican en tres tipos:

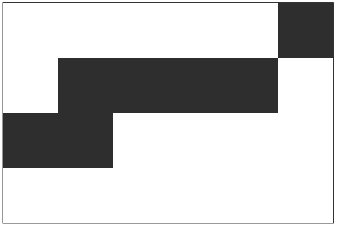
- Fuertes: pixeles con magnitud del gradiente mayor que el umbral superior $|\nabla| > \theta_u$
- Débiles: pixeles con magnitud del gradiente entre ambos umbrales $\theta_l < |\nabla| < \theta_u$
- Suprimidos: pixeles con magnitud del gradiente menor que el umbral inferior $|\nabla| < \theta_l$

La imagen resultante se forma con las siguientes reglas:

- Todos los pixeles fuertes son parte de un borde.
- Todos los pixeles suprimidos no son bordes.
- Los pixeles débiles son parte de un borde solo si están conectados (en conectividad 8) con un pixel fuerte.







Espacios de color

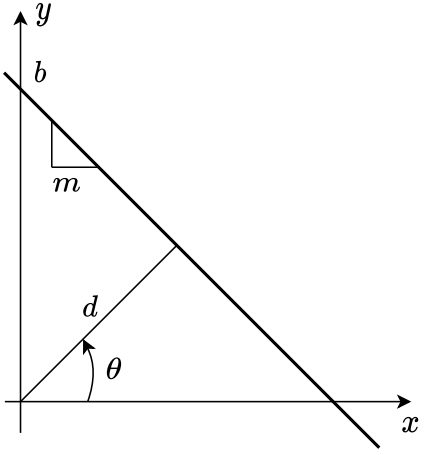
En las imágenes el color se representa mediante la combinación de 3 valores (generalmente, 3 bytes). Las diferentes formas de representar el color mediante estos tres valores se conocen como *espacios de color*.

Extracción de características

- Las *características* (*features*) son conjuntos de valores derivados de una señal o de un conjunto de mediciones, que permiten describir el conjunto original y que son informativos y no redundantes.
- Unas de las características más simples que se pueden extraer de una imagen son líneas y otras formas geométricas simples.
- El vector de características que describe una línea puede ser el conjunto de parámetros de su ecuación.
- Los vectores de características también se pueden utilizar para reconocer objetos, usando características cmo color, tamaño, forma, volumen, entre otras.
- Más adelante se verá la Transformada SIFT, uno de los algoritmos más usados para extracción de características.

La Transformada Hough

La Transformada Hough es un método que permite encontrar líneas, círculos y otras formas geométricas que se puedan describir fácilmente mediante expresiones analíticas. En el caso de las líneas, se trata de encontrar los dos parámetros que describen la recta:

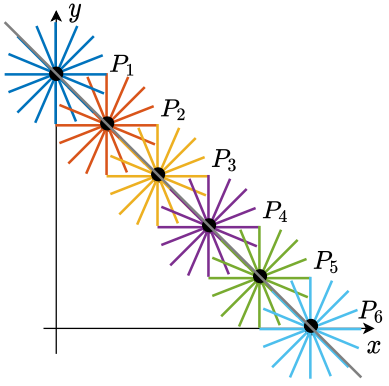


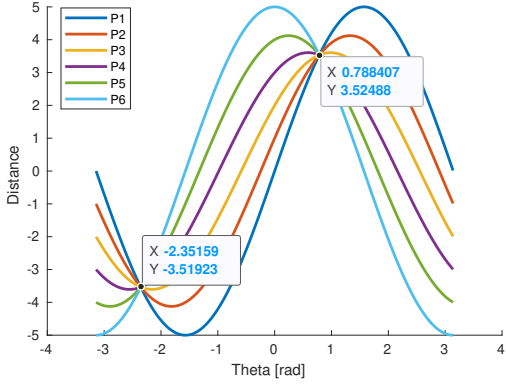
- La forma pendiente-ordenada $y = mx + b$ tiene la desventaja de no poder expresar líneas verticales.
- La forma canónica $Ax + By + C$ requiere de una normalización $A_1x + B_1y + 1 = 0$ para que solo sean dos parámetros.
- Una forma más conveniente, es la forma normal $d = x\cos\theta + y\sin\theta$
- Esta última forma tiene una ventaja: si la línea correponde a un borde, el ángulo θ será la dirección del gradiente.

El Espacio de Hough

El Espacio de Hough, para el caso de las líneas, es el conjunto de todos los posibles pares (θ, d).

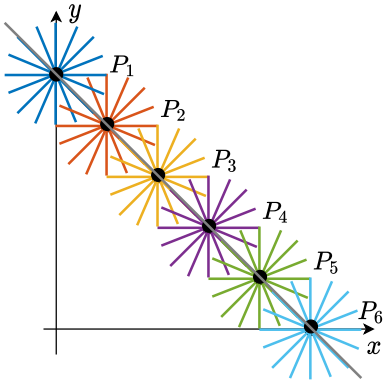
- Una recta L en el espacio cartesiano corresponde a un punto P_h en el Espacio de Hough
- Un punto P_c en el espacio cartesiano corresponde a una curva C en el Espacio de Hough. Esta curva representa los parámetros (θ_i, d_i) de todas las rectas L_i que pasan por el punto P.

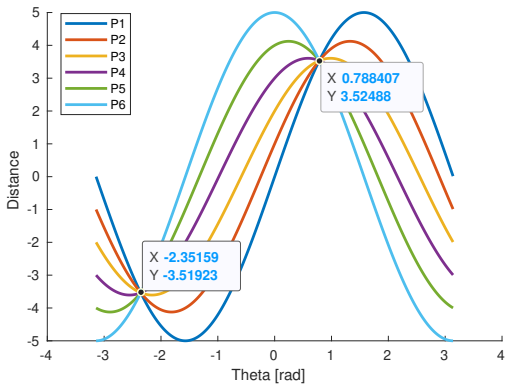




Extracción de Líneas por Transformada Hough

Este método consiste en encontrar las curvas C_i en el espacio de Hough que pasan por cada punto P_c en el espacio cartesiano. Los puntos P_h en el Espacio de Hough por donde pasen más curvas C_i corresponderán a las rectas resultantes en el espacio cartesiano.





Extracción de Líneas por Transformada Hough

Input: Imagen binaria M, umbral mínimo de votos a_{min}
Output: Líneas expresadas en la forma (d, θ)

Inicializar en 0 un conjunto A de acumuladores para cada par cuantizado (d_k, θ_k)
forall *Pixel* $M[i, j] \neq 0$ **do**
  forall *Ángulo* θ_k *cuantizdo* **do**
    $d = j \cos\theta_k + i \sin\theta_k$ $d_k =$ valor cuantizado de d Incrementar en uno el acumulador correspondiente $A[d_k, \theta_k]$
  end
end
forall $a \in A$ **do**
  if $a > a_{min}$ **then**
    Agregar la línea (d_k, θ_k) al conjunto de líneas detectadas
  end
end
Devolver el conjunto de líneas detectadas

Extracción de círculos por T. Hough

De la ecuación de un círculo en el plano:

$$(x - c_x)^2 + (y - c_y)^2 = r^2$$

Se puede observar que un círculo está definido por tres parámetros (c_x, c_y, r).

- De forma similar a las líneas, se podrían fijar dos parámetros y calcular el tercero
- Sin embargo variar dos parámetros incrementa la complejidad considerablemente
- Se puede utilizar el ángulo del gradiente y un variar el valor del radio r para determinar el centro, de este modo, solo se tiene que variar un parámetro.
- Se considera que el gradiente apunta hacia el centro del círculo.

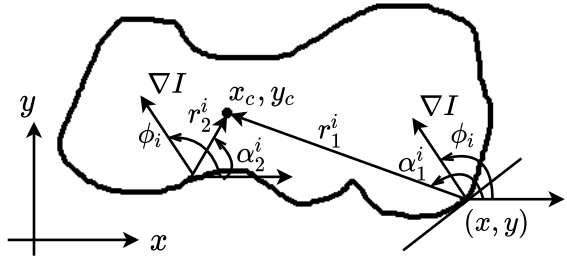
T. Hough Generalizada

Hasta el momento la T. Hough se ha usado para detectar formas determinadas analíticamente. Se pueden detectar formas arbitrarias si se dispone de una descripción de la forma a detectar y, al igual que con círculos y líneas, se varían los parámetros y se hace un sistema de votos.

- La descripción de la forma se hace mediante la llamada *Tabla R*
- Los parámetros que se pueden variar son el centro (x_c, y_c), la rotación θ y la escala (S_x, S_y)

T. H. Generalizada - Tabla R

La Tabla R es una descripción de la figura en función del ángulo del gradiente.



- Se selecciona un centro arbitrario (x_c, y_c)
- Para cada punto (x, y) en la figura, se calcula el ángulo del gradiente y se cuantiza a un ángulo ϕ_i.
- Para el mismo punto, se calcula el vector del punto al centro $(x_c - x, y_c - y)$ que se puede expresar en la forma polar (r, α)
- A cada ángulo cuantizado ϕ_i se le pueden asociar varios valores $(r_1^i, \alpha_1^i), \ldots, (r_j^i, \alpha_j^i)$

T. H. Generalizada - Tabla R

Con la información anterior se construye la Tabla R:

Ángulo del Gradiente	Vector al centro
ϕ_1	$(r_1^1, \alpha_1^1), \ldots, (r_j^1, \alpha_j^1)$
ϕ_2	$(r_1^2, \alpha_1^2), \ldots, (r_j^2, \alpha_j^2)$
$\vdots$	$\vdots$
ϕ_i	$(r_1^i, \alpha_1^i), \ldots, (r_j^i, \alpha_j^i)$
$\vdots$	$\vdots$
ϕ_n	$(r_1^n, \alpha_1^n), \ldots, (r_j^n, \alpha_j^n)$

La Tabla R funciona como una descripción de la forma arbitraria.

Detección de formas generales

El caso más sencillo se da cuando se tiene una forma en la imagen del mismo tamaño y orientación que el patrón original. En este caso solo es necesario determinar el centro, por lo que el Espacio de Hough solo tiene dos dimensiones: los parámetros x_c, y_c.

Input: Imagen binaria M, umbral mínimo de votos a_{min}
Output: Formas generales definidas por la tabla R con centro en (x_c, y_c)

Inicializar en 0 un conjunto A de acumuladores para cada par cuantizado (x_c, y_c)
forall *Pixel* $M[x, y] \neq 0$ **do**
    Determinar el ángulo ϕ del gradiente en el punto x, y y cuantizarlo a un valor ϕ_i
    forall *Vector* (r_i^j, α_i^j) *asociado al ángulo* ϕ_i *en la Tabla R* **do**
        Determinar las coordenadas del centro:
        $x_c^j = x + \cos \alpha_i^j$
        $y_c^j = y + \sin \alpha_i^j$ Incrementar en uno el acumulador correspondiente $A[x_c^j, y_c^j]$
    end
end
forall $a \in A$ **do**
    if $a > a_{min}$ **then**
        Agregar el patrón con centro en (x_c, y_c) al conjunto de patrones detectados
    end
end
Devolver el conjunto de patrones detectados

Detección de formas generales

Para detectar patrones con orientación θ y escala S arbitrarios, se aplica el mismo principio, pero con un espacio de Hough de cuatro dimensiones: $(x_c, y_c, \theta, S$

Input: Imagen binaria M, umbral mínimo de votos a_{min}
Output: Formas generales definidas por la tabla R

Inicializar en 0 un conjunto A de acumuladores para cada tupla cuantizada (x_c, y_c, θ, S)
forall *Pixel* $M[x, y] \neq 0$ **do**
  Determinar el ángulo ϕ del gradiente en el punto x, y y cuantizarlo a un valor ϕ_i
  forall *Orientación cuantizada* θ **do**
    forall *Escala cuantizada* S **do**
      forall *Vector* (r_i^j, α_i^j) *asociado al ángulo* ϕ_i *en la Tabla R* **do**
        Determinar las coordenadas del centro:
        $x_c^j = x + S\cos(\alpha_i^j + \theta)$
        $y_c^j = y + S\sin(\alpha_i^j + \theta)$
        Incrementar en uno el acumulador correspondiente $A[x_c^j, y_c^j, \theta, S]$
      end
    end
  end
end
Devolver el conjunto de patrones donde el acumulador $A[x_c^j, y_c^j, \theta, S] > a_{min}$

Detección de esquinas

Una esquina se puede considerar como un punto donde, en una vecindad W, el gradiente tiene dos direcciones dominantes diferentes.

- Aunque las esquinas representan un porcentaje muy pequeño de la imagen, suelen ser puntos de mucho interés.
- Las esquinas se consideran puntos de interés invariantes a escala, rotación e iluminación
- Entre algunas aplicaciones se encuentran:
  - Registro de imágenes
  - Reconstrucción de escenas 3D
  - Detección de movimiento
  - Reconocimiento de objetos

Detector de esquinas de Harris

Este detector se basa en la autocorrelación de los valores del gradiente en una vencidad de un pixel (x, y). La detección se puede sintetizar en los siguientes pasos:

- Obtención de las derivadas parciales
- Obtención de la matriz del segundo momento
- Cálculo de la respuesta de Harris
- Supresión de no máximos

Laplaciano de Gaussiano (LoG)

El Laplaciano de una función está definido como:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Aplicando diferencias finitas, el Laplaciano se puede aproximar con el Kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$