Tarea N° 1

Aprendiendo a Ver

Cristian Herrera Badilla
Escuela de Ingeniería, Universidad de O'Higgins
Sábado 27, Abril, 2024

***Abstract*—Este documento explora dos áreas clave en informática y robótica: el procesamiento de imágenes con OpenCV y la simulación de entornos de aprendizaje por refuerzo con Gym-Duckietown. En la sección del Marco Teórico, se introducen los conceptos fundamentales de OpenCV y se detalla su aplicación en diversas tareas de procesamiento de imágenes. Además, se presenta una visión general de Gym-Duckietown y su utilidad en la investigación en robótica y aprendizaje por refuerzo. En la sección de Metodología, se describen los métodos utilizados para experimentar en ambos campos. Finalmente, en la sección de Resultados y Análisis, se presentan y analizan los resultados obtenidos, destacando la versatilidad y eficacia de las técnicas y algoritmos aplicados.**

I. Introducción

El presente documento tiene como objetivo explorar y analizar dos áreas fundamentales en el campo de la robótica: el procesamiento de imágenes utilizando la biblioteca OpenCV y la simulación de entornos de aprendizaje por refuerzo mediante Gym-Duckietown.

Se proporciona una visión general de OpenCV, una potente biblioteca de código abierto diseñada para el procesamiento de imágenes y la visión por computadora. Se presenta una introducción al entorno de simulación Gym-Duckietown, que es utilizado para la investigación en robótica y aprendizaje por refuerzo.

Se detallan los métodos utilizados para realizar experimentos y análisis. En particular, se detalla el uso de Google Colab para el procesamiento de imágenes con OpenCV, incluyendo la aplicación de diferentes filtros y la comparación de espectros mediante la transformada de Fourier rápida (FFT). También se presenta la implementación de algoritmos para la detección y seguimiento de objetos en la simulación de Gym-Duckietown, haciendo uso de técnicas de procesamiento de imágenes por color y operaciones morfológicas.

II. Marco teórico

A. OpenCV

Open Source Computer Vision Library, es una biblioteca de software de código abierto diseñada para el procesamiento de imágenes y la visión por computadora. Proporciona una amplia gama de funciones y algoritmos que permiten trabajar con imágenes y videos de manera eficiente.

1) Procesamiento de imágenes: Gran variedad de funciones para cargar, manipular, procesar y analizar imágenes digitales. Esto incluye operaciones básicas como el cambio de tamaño, la rotación, la conversión de colores, la detección de bordes, la segmentación y la eliminación de ruido, entre otras.
2) Detección y seguimiento de objetos: Algoritmos para la detección y seguimiento de objetos en imágenes y videos, incluidos métodos basados en características, como descriptores de características y correspondencia de características, así como técnicas de aprendizaje profundo.
3) Reconocimiento facial: Herramientas para el reconocimiento facial, que permiten detectar y reconocer rostros en imágenes y videos, así como realizar tareas como la detección de expresiones faciales y el seguimiento de rostros en tiempo real.
4) Visión estéreo y reconstrucción 3D: Funcionalidades para trabajar con imágenes estéreo y realizar reconstrucción tridimensional a partir de múltiples vistas de una escena.
5) Aprendizaje automático y aprendizaje profundo: Integración con bibliotecas populares de aprendizaje automático y aprendizaje profundo, como TensorFlow y PyTorch.

B. Convolución

Función fundamental en el procesamiento de imágenes digitales, la cual realiza el filtrado de los valores de píxel de una imagen, lo que se puede utilizar para aumentar su nitidez, difuminarla, detectar sus ejes u otros realces basados en el kernel. En Python se realiza mediante bibliotecas como NumPy y OpenCV.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau$$

Esta fórmula representa la convolución de dos funciones f y g, donde $*$ es el operador de convolución y $(\cdot)$ denota la multiplicación de funciones. El resultado de la convolución se obtiene integrando el producto de las dos funciones trasladadas una respecto a la otra a lo largo del eje τ.

En el contexto del procesamiento de imágenes discretas, esta fórmula se simplifica a una sumatoria sobre los índices de las matrices que representan las imágenes. La convolución discreta se realiza multiplicando elemento por elemento la imagen de entrada con el kernel y sumando los productos correspondientes.

C. kernel

Matriz cuadrada de números que actúa como una especie de "filtro" o "máscara" que se aplica a cada píxel de la imagen

de entrada durante la convolución. Esta matriz puede tener diferentes tamaños, como 3x3, 5x5 o incluso más grande, y sus valores determinan cómo se realizará la transformación en la imagen.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Fig. 1: Sharpen

D. Gym-Duckietown

Gym-Duckietown es un entorno de simulación de código abierto diseñado para la investigación en robótica y aprendizaje por refuerzo. Se basa en la plataforma Gym de OpenAI, que proporciona un conjunto de entornos para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Este entorno simula Duckietown, un proyecto educativo de robótica centrado en la construcción de pequeños vehículos autónomos llamados "Duckiebots" que se mueven sobre pistas modulares.

1) Interfaz Gym: Al estar basado en Gym, sigue la estructura estándar de los entornos de Gym, lo que facilita la integración con algoritmos de aprendizaje por refuerzo. Proporciona una interfaz común que permite a los desarrolladores interactuar con el entorno, tomar acciones y recibir observaciones y recompensas del entorno.
2) Tareas de aprendizaje: En Gym-Duckietown, los usuarios pueden definir una variedad de tareas de aprendizaje para que los Duckiebots las aborden. Estas tareas pueden incluir la navegación autónoma en un entorno desconocido, seguir una ruta específica, evitar obstáculos o incluso interactuar con otros robots en el entorno.

III. Metodología

A. Google Colab

Primero se importan las librerías cv2, numpy, matplotlib.pyplot y cv2_imshow para utilizar la herramienta de opneCV convolución (función: cv2.filter2D) y crear matrices que serán nuestros "kernel" para realizar los diferentes filtros a nuestras imagenes. Las imágenes se obtienen de un repositorio de github con la función !wget y se transforman a escala de grises y se asigna cada una a una variable utilizando cv2.imread y cv2.IMREAD_GRAYSCALE.

Código 1: Función convolución

```
def convolution(gray_image, mask):
    output_image = cv2.filter2D(gray_image, -1, mask)
    return output_image
```

Para cada filtro se hizo uso de distintas matrices como la siguiente:

Código 2: kernel

```
mask = np.array([[1, 1, 1],
                 [1, 1, 1],
                 [1, 1, 1]])*(1/9)
```

Para el filtro gaussiano se uso la función cv2.GaussianBlur. Y finalmente para el análisis y comparación de espectros se utilizo FFT.

Código 3: FFT

```
outputs = [output_image_recto, output_image_unidimensional,
      output_image_Gaussiano_bidimensional]
titulos = ["Filtro pasa bajos recto", "Filtro pasa bajos
     unidimensional", "Filtro Gaussiano pasa bajos
     bidimensional"]
for output in zip(outputs, titulos):
  f_transform = np.fft.fft2(output[0])
  f_transform_shifted = np.fft.fftshift(f_transform)
  magnitude_spectrum = 20 * np.log(np.abs(
     f_transform_shifted))
```

B. Gym-Duckietown

A partir del script "manual_control.py" se creo dos funciones. Una para la creación de una nueva "ventana" donde se detecta al patito, esta vista dibuja un rectángulo sobre el con su etiqueta "Duckie" y en base al tamaño de dicho rectángulo se activa un freno de emergencia que evita atropellar al pato, así llamando a la segunda función vuelve la vista roja y da aviso del frenado de emergencia.

Código 4: Detector de patitos

```
def Duck_detector(image):
    # Set minimum and max HSV values to display
    lower = np.array([94, 253, 176])
    upper = np.array([103, 255, 255])

    # Create HSV Image and threshold into a range.
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv, lower, upper)

    # Do perform morphological operations
    kernel = np.ones((5,5),np.uint8)
    op_morf = cv2.erode(mask,kernel,iterations = 1)
    op_morf = cv2.dilate(op_morf,kernel,iterations = 20)

    # To create contours.
    contours, hierarchy = cv2.findContours(op_morf, cv2.
     RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for c in contours:
        x,y,w,h = cv2.boundingRect(c)
        cv2.rectangle(image, (x,y), (x + w, y + h),
     (0,0,255), 2)

        # Emergency brake
        if w > 300 or h > 300:
            action = np.array([-0.44, 0])
            env.step(action)
            image = red_alert(image)
            cv2.putText(image, 'Freno de Emergencia!',
     (150, 100) , cv2.FONT_HERSHEY_SIMPLEX , 1, (0,0,255),
     2, cv2.LINE_AA)

        else:
            cv2.putText(image, 'Duckie', (x,y - 10) , cv2.
     FONT_HERSHEY_SIMPLEX , 1, (0,0,255), 2, cv2.LINE_AA)
    return image
```

Aquí se hace uso de dos mascaras las cuales están representadas en la siguiente tabla.

Canal	Min.	Max.
H	94	103
S	253	255
V	176	255

TABLE I: Mascaras

Estos parámetros fueron encontrados en base a prueba y error con el objetivo de filtrar por el color amarillo y naranja (colores representativos del patito). También en base a prueba y error se llego a los siguientes valores de erosión y dilatación (1 y 20 respectivamente). De esta forma gracias a la erosión se pueden eliminar errores de detección, por ejemplo así se evita detectar las luces de dirección de los vehículos.

Código 5: Alerta

```
def red_alert(frame):
    red_img = np.zeros((480, 640, 3), dtype = np.uint8)
    red_img[:,:,2] = 90
    blend = cv2.addWeighted(frame, 0.5, red_img, 0.5, 0)

    return blend
```

IV. Resultados

A. *Google Colab*

Algunos resultados del filtrado de la imagen "calle1.png" y el espectro de la imagen "calle6.png"

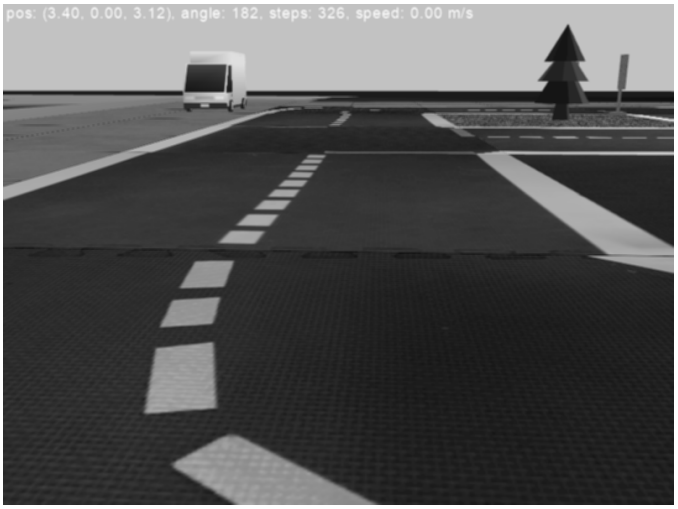

Fig. 2: Filtro pasa-bajos recto

Fig. 3: Filtro pasa-altos Prewitt vertical

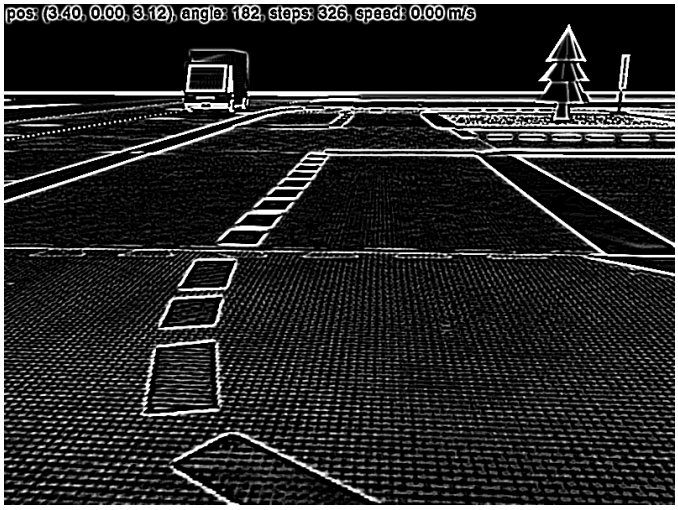

Fig. 4: Filtro laplaciano de gaussiana de 5x5

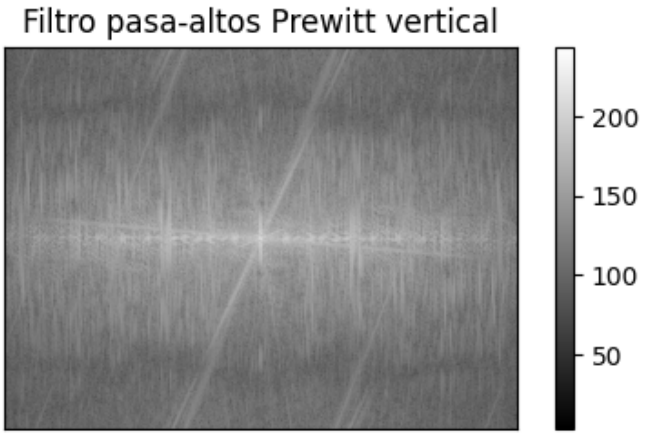

Fig. 5: Filtro pasa-altos Prewitt vertical

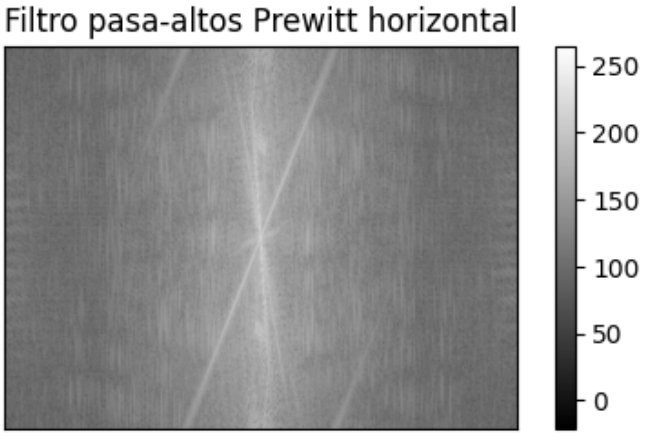

Fig. 6: Filtro pasa-altos Prewitt horizontal

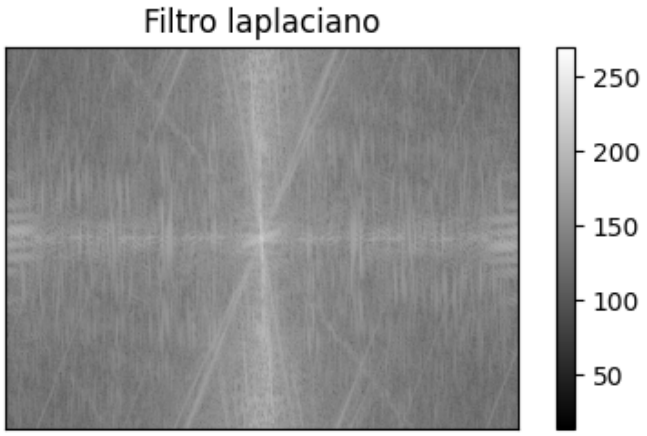

Fig. 7: Filtro laplaciano

Fig. 8: Filtro laplaciano de gaussiana

B. *Gym-Duckietown*

Captura del resultado en la simulación.

Fig. 9: Detectando al patito a lo lejos

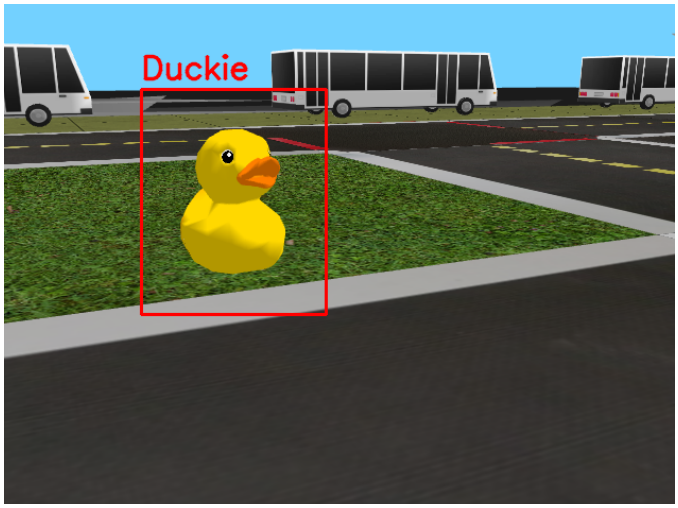

Fig. 10: Detectando al patito de cerca

Fig. 11: Frenado de emergencia

V. Análisis

A. *Google Colab*

Se puede observar que algunas de las imágenes filtradas con los diferentes filtros se ven semejantes a otras, un ejemplo claro son las imágenes usando el filtrado pasa-bajos recto y el filtrado pasa-bajos unidimensional. No es el caso de los filtrados gaussianos ya que aquí es posible notar las imágenes difuminadas. Sin embargo donde se puede apreciar de mejor manera las diferencias es usando la función FFT, la cual muestra los espectro de magnitud que hay en la imagen dando a conocer a que frecuencias afecta cada filtro.
El espectro del filtro pasa bajos-recto es mas parecido al filtro gaussiano pasa-bajos bidimensional ya que en ambos las esquinas del espectro son mas oscuras y hay una especie de cruz en el centro. En cambio el espectro del filtro pasa-bajos unidimensional tiene dos lineas horizontales paralelas. Así mismo los filtro pasa-altos Prewitt vertical y pasa-altos Prewitt horizontal muestran un espectro con lineas horizontales para el primer filtro mencionado y verticales para el segundo. En este ultimo analisis se puede ver la gran diferencia que se

crea en el espectro gaussiano usando el filtrado laplaciano y esto es algo bastante curioso ya que el espectro del filtrado laplaciano por si solo no muestra patrones reconocibles.

B. Gym-Duckietown

Para detectar al duckie dentro de la simulación se utilizo un filtrado por color, en este caso amarillo y naranja, los problemas que suponía esta elección eran los demás objetos con estos mismos colores. Por ende se utilizo las operaciones morfológicas erosión y dilatación para contrarrestar estas falsas detecciones. Para crear el freno de mano se utiliza una decisión que se activa con los valores de altura y largo del rectángulo detector, y como no funciona asignar la velocidad en 0 se opta por utilizar la misma velocidad que se usa para avanzar pero negativa.

VI. Conclusiones generales

A lo largo de este estudio, se ha demostrado la versatilidad y eficacia de OpenCV en una amplia gama de tareas de procesamiento de imágenes, desde la detección de bordes hasta la segmentación y el reconocimiento de objetos.

Los resultados y análisis presentados han revelado la capacidad de las técnicas y algoritmos aplicados para abordar desafíos específicos en cada área. En el procesamiento de imágenes, se observaron diferencias significativas entre los diversos filtros aplicados, especialmente al analizar los espectros de frecuencia utilizando la transformada de Fourier (FFT). Por otro lado, en la simulación de Gym-Duckietown, se implementaron con éxito algoritmos de detección y seguimiento de objetos utilizando técnicas de procesamiento de imágenes por color y operaciones morfológicas.

Este estudio proporciona una comprensión profunda de cómo OpenCV y Gym-Duckietown pueden ser utilizados en aplicaciones prácticas y de investigación en informática y robótica. Estas herramientas y técnicas ofrecen un amplio potencial para el desarrollo de sistemas inteligentes y autónomos en una variedad de campos.