Carrera: Licenciatura en Sistemas.

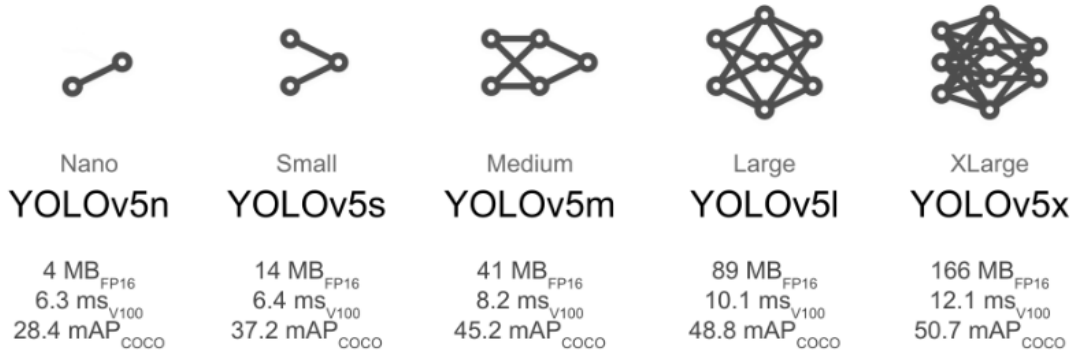
Curso: Python y Robótica II.

Tema: Actividad n°3.

Integrantes: Costas Alexis Maximiliano, Dri Alán David, Garcia Gelsi Ramiro, Horta Fernando.

Fecha de Entrega: 01/11/2023

En un principio decidimos realizar la actividad propuesta mediante el uso de redes neuronales, en la busqueda de informacion encontramos un algoritmo de Haar Cascade (basado en aprendizaje en cascada), que nos permite entrenar una red alimentandola con imágenes positivas, que en nuestro caso serían los carteles que tenemos que detectar y las imágenes negativas, que sería la contracara de las positivas, donde los carteles no están presentes, para estas utilizamos fotos tomadas con las raspberry del fondo donde íbamos a realizar la prueba de la actividad.
Una vez entrenado este algoritmo nos devuelve un archivo ".yaml" el cual levantamos con python para el reconocimiento de las entidades. El resultado de esta prueba fue malo, ya que no detectaba ninguna señal.
Luego seguimos buscando información y encontramos la red YOLOv5 que tiene varias redes, en un principio utilizamos la red más compleja que es la que más puntos de interconexión tiene que es la YOLOv5x.



Entrenamos nuestro propio set de imágenes que eran las fotos tomadas de los carteles tanto tomadas con la raspberry, como con un celular y algunas descargadas de internet. Utilizamos el software makesense.ai para recortar nuestras imagenes y clasificarlas según el tipo de señal, al final de esta etapa se exporta nuestro dataset personalizado con las imagenes para entrenar nuestra red (train) y sus correspondientes archivos ".txt" con los datos de las imágenes para entrenar nuestra red. Se repite el mismo proceso para las imagen de validación (val).
Luego en el google colab que nos provee el propio github del proyecto cargamos nuestros datos y los entrenamos, con un número de 150 épocas de entrenamiento y utilizando la red YOLOv5x.
Al finalizar esta etapa exportamos un archivo ".pt" que es nuestra red entrenada, la cargamos utilizando python y la librería pytorch obteniendo un resultado satisfactorio, ya que reconoció las imágenes.
El problema surgió a la hora de ejecutar este script en la raspberry pi 4 ya que las pruebas las hicimos en una notebook. Al momento de ejecutar este script en la raspy nos arrojaba un error que ocasiona un problema al momento de fusionar las capas y termina abortando la ejecución. Buscando dicho error por internet encontramos que podría referirse a que el modelo era muy pesado y por ello no lo podía levantar la raspberry, por lo que se optó por repetir el proceso anterior pero utilizando el modelo YOLOv5s y entrenar solamente con 100 épocas.
Una vez obtenido el archivo ".pt" lo probamos en la notebook y funcionaba perfecto pero cuando lo llevamos a la raspberry pi 4 nos arrojaba el mismo error anterior.

Se siguió buscando información, se modificó el tamaño del swap del sistema que podría ser el causante del error anterior pero tampoco lo soluciono.
Por último, decidimos seguir intentando mediante el reconocimiento de contornos y el uso de un algoritmo de comparación de fuerza bruta con opencv que es el que se muestra a continuación y el cual presentamos el día martes 31/10/2023.
Este algoritmo funciona, teniendo algunos inconvenientes con el cartel de izquierda y derecha, para el cual probamos utilizar el algoritmo dos veces y si lo reconoce igual en las esas instancias se confirma el cartel, pero este proceso hace que el robot demore unos segundo al momento de ejecutar la detección ya que es un proceso más lento.
Para solucionar este inconveniente decidimos acercarnos a los carteles utilizando la detección por colores, y que solo ejecute el algoritmo de fuerza bruta una vez que detecta que el contorno detectado ocupa el 40% de la captura de la cámara.
A continuación se presenta el algoritmo utilizado:

```
import numpy as np
import cv2
import math
import RPi.GPIO as GPIO
from time import sleep
import time

def arranque():
    GPIO.setmode (GPIO.BOARD)
    GPIO.setup (11,GPIO.OUT)
    GPIO.setup (13,GPIO.OUT)
    GPIO.setup (15,GPIO.OUT)
    GPIO.setup (16,GPIO.OUT)

def liberar_recursos(inicio = True):
    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(16,False)
    GPIO.output(15,False)
    if (inicio == False):
        GPIO.cleanup()

def forward(tiempo = 2):
    GPIO.output(11,GPIO.LOW)
    GPIO.output(13,GPIO.HIGH)
    GPIO.output(16,GPIO.HIGH)
    GPIO.output(15,GPIO.LOW)
    time.sleep(tiempo)
    liberar_recursos()
```

```
def reverse(tiempo = 2):
    GPIO.output(11,GPIO.HIGH)
    GPIO.output(13,GPIO.LOW)
    GPIO.output(16,GPIO.LOW)
    GPIO.output(15,GPIO.HIGH)
    time.sleep(tiempo)
    liberar_recursos()

def turn_left(tiempo = 1):
    GPIO.output(11,GPIO.LOW)
    GPIO.output(13,GPIO.HIGH)
    GPIO.output(16,False)
    GPIO.output(15,False)
    time.sleep(tiempo)
    liberar_recursos()

def turn_right(tiempo = 1):
    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(16,GPIO.HIGH)
    GPIO.output(15,GPIO.LOW)
    time.sleep(tiempo)
    liberar_recursos()

def giro_90_izq(tiempo = 1.5):
    # reverse
    # GPIO.output(11,GPIO.HIGH)
    # GPIO.output(13,GPIO.LOW)
    # GPIO.output(16,GPIO.LOW)
    # GPIO.output(15,GPIO.HIGH)
    # time.sleep(tiempo - 0.7)
    # left
    GPIO.output(11,GPIO.LOW)
    GPIO.output(13,GPIO.HIGH)
    GPIO.output(16,False)
    GPIO.output(15,False)
    time.sleep(tiempo)

    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(16,False)
    GPIO.output(15,False)
```

```
def giro_90_der(tiempo = 1.5):
    # reverse
    # GPIO.output(11,GPIO.HIGH)
    # GPIO.output(13,GPIO.LOW)
    # GPIO.output(16,GPIO.LOW)
    # GPIO.output(15,GPIO.HIGH)
    # time.sleep(tiempo - 0.7)
    # right
    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(16,GPIO.HIGH)
    GPIO.output(15,GPIO.LOW)
    time.sleep(tiempo)

    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(16,False)
    GPIO.output(15,False)

def giro_130_izq():
    reverse(0.8)
    turn_left(2)

def vuelta_entera():
    reverse(1)
    turn_right(4)
    print("¡¡¡festejo!!!")
    azul_completado = True

def accion_pare():
    time.sleep(5)
    giro_90_izq()
    forward(0.5)
    giro_90_der()
    forward(0.5)
    liberar_recursos()

def accion_giro_u():
    reverse(0.8)
    turn_left(4)
    forward(1)

def accion_adelante():
```

```
    giro_90_izq()
    forward(0.5)
    giro_90_der()
    forward(0.5)
    liberar_recursos()

def encontrar(grey_frame, imageToCompare):
    #Already tested resize the image outside the function to reduce
time but it does the oppose
    nuevo_ancho = frame.shape[1] // 2
    nuevo_alto = frame.shape[0] // 2
    frame_gris_redimensionada = cv2.resize(grey_frame, (nuevo_ancho,
nuevo_alto))
    # Encontrar puntos clave y descriptores del frame
    keypointsFrame, descriptorsFrame =
sift.detectAndCompute(frame_gris_redimensionada, None)
    try:
        if (imageToCompare == 0):
            matches = bf.knnMatch(descr_pare, descriptorsFrame, k=2)
        elif (imageToCompare == 1):
            matches = bf.knnMatch(descr_adelante, descriptorsFrame,
k=2)
        elif (imageToCompare == 2):
            matches = bf.knnMatch(descr_izquierda, descriptorsFrame,
k=2)
        elif (imageToCompare == 3):
            matches = bf.knnMatch(descr_derecha, descriptorsFrame, k=2)
        elif (imageToCompare == 4):
            matches = bf.knnMatch(descr_prohibido_u, descriptorsFrame,
k=2)
    except:
        return 7
    # Configurar el algoritmo de correspondencia

    # Aplicar la razón de Lowe para filtrar las correspondencias
    good_matches = []
    try:
        for m, n in matches:
            if m.distance < umbral * n.distance:
                good_matches.append(m)
    except:
        reverse(0.1)
    # Si hay suficientes correspondencias, encontrar la señal detectada
```

```
    return len(good_matches)

def definirSignal(grey_frame):
    matches = []
    nombreDeMatch = []
    if not pare_found:
        matches.append(encontrar(grey_frame,0))
        nombreDeMatch.append("pare")
    if not adelante_found:
        matches.append(encontrar(grey_frame,1))
        nombreDeMatch.append("adelante")
    if not izquierda_found:
        matches.append(encontrar(grey_frame,2))
        nombreDeMatch.append("izquierda")
    if not derecha_found:
        matches.append(encontrar(grey_frame,3))
        nombreDeMatch.append("derecha")
    if not prohibido_u_found:
        matches.append(encontrar(grey_frame,4))
        nombreDeMatch.append("u")

    imagen_encontrada=''
    max_matchs = 10
    i=0
    # print("#######################################")
    # print("Comienzo de procesado de imagen")
    for match in matches:
        # print("El match para ")
        # print(nombreDeMatch[i])
        # print(matches[i])
        if matches[i] > max_matchs:
            max_matchs = matches[i]
            imagen_encontrada = nombreDeMatch[i]

        i+=1

    # print("la imagen es: ", imagen_encontrada)
    # print("el match mas grande es: ", max_matchs)
    if (imagen_encontrada == "pare"):
        return 0
    if (imagen_encontrada == "adelante"):
        return 1
    if (imagen_encontrada == "izquierda"):
```

```
        return 2
    if (imagen_encontrada == "derecha"):
        return 3
    if (imagen_encontrada == "u"):
        return 4

def afirmarSignal(imagen_encontrada):
    ret, frame = cap.read()
    num_imagen_para_afirmar = definirSignal(frame)
    if (imagen_encontrada == num_imagen_para_afirmar):
        return True
    else:
        return False

arranque()

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

azul_completado = False
color_actual = 'Rojo'
area_prom = 0
contador = 0
busqueda_iniciada = False
empezar = True
cont = 0
area_g = 0.2
k = cv2.waitKey(1)
j = True

imagen_pare = cv2.imread('img/pare.jpg', 1)
imagen_adelante = cv2.imread('img/recto.jpg', 1)
imagen_balneario = cv2.imread('img/balneario.jpg', 1)
imagen_izquierda = cv2.imread('img/izquierda.jpg', 1)
imagen_derecha = cv2.imread('img/derecha.jpg', 1)
imagen_prohibido_u = cv2.imread('img/u.jpg', 1)

sift = cv2.SIFT_create()
umbral = 0.7
bf = cv2.BFMatcher()

#Keypoints and descriptors
```

```
kp_pare, descr_pare = sift.detectAndCompute(imagen_pare, None)
kp_adelante, descr_adelante = sift.detectAndCompute(imagen_adelante,
None)
kp_balneario, descr_balneario = sift.detectAndCompute(imagen_balneario,
None)
kp_izquierda, descr_izquierda = sift.detectAndCompute(imagen_izquierda,
None)
kp_derecha, descr_derecha = sift.detectAndCompute(imagen_derecha, None)
kp_prohibido_u, descr_prohibido_u =
sift.detectAndCompute(imagen_prohibido_u, None)

pare_found = False
adelante_found = False
balneario_found = False
izquierda_found = False
derecha_found = False
prohibido_u_found = False

while j:
    z = input("Ingrese s para saludar ")
    if z == "s":
        print("arrancar")
        forward(1)
        reverse(1)
        j = False

j = True
z = ""

while j:
    z = input("Ingrese s para empezar ")
    if z == "s":
        j = False

while not azul_completado:
    ret, frame = cap.read()
    if not ret:
        break

    if empezar:
        cv2.imshow("Video", frame)

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

```
    # Define los rangos de color para los colores a detectar
    if color_actual == "Rojo":

        # Definir los umbrales para el rojo brillante
        low_red1 = np.array([0, 125, 150])
        high_red1 = np.array([15, 255, 255])

        # Definir los umbrales para el rojo pálido o lavado
        low_red2 = np.array([160, 75, 150])
        high_red2 = np.array([180, 255, 255])

        # Convertir el color RGB en formato HSV
        rgb_color = np.uint8([[[162, 100, 108]]])
        hsv_color = cv2.cvtColor(rgb_color, cv2.COLOR_RGB2HSV)

        # Definir un rango de tolerancia para el color en formato HSV
        lower_bound = np.array([hsv_color[0][0][0] - 10, 50, 50])
        upper_bound = np.array([hsv_color[0][0][0] + 10, 255, 255])

        # Convertir la imagen a formato HSV
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Crear las máscaras
        mask_color = cv2.inRange(hsv, lower_bound, upper_bound)
        mask_red1 = cv2.inRange(hsv, low_red1, high_red1)
        mask_red2 = cv2.inRange(hsv, low_red2, high_red2)

        # Combina todas las máscaras
        mask = cv2.bitwise_or(mask_color, cv2.bitwise_or(mask_red1,
mask_red2))

        # # Definir los umbrales para el blanco
        # low_white = np.array([0, 0, 200])
        # high_white = np.array([255, 30, 255])

        # # Crear las máscaras
        # mask_color = cv2.inRange(hsv, lower_bound, upper_bound)
        # mask_red1 = cv2.inRange(hsv, low_red1, high_red1)
        # mask_red2 = cv2.inRange(hsv, low_red2, high_red2)
        # mask_white = cv2.inRange(hsv, low_white, high_white)
```

```
        # # Combinar todas las máscaras
        # mask = cv2.bitwise_or(mask_color, cv2.bitwise_or(mask_red1,
cv2.bitwise_or(mask_red2, mask_white)))

    elif color_actual == "Azul":
        low_blue = np.array([95, 100, 20])
        high_blue = np.array([135, 255, 255])
        mask = cv2.inRange(hsv, low_blue, high_blue)


    # suavizado y operaciones de erosión y dilatación
    mask = cv2.GaussianBlur(mask, (5, 5), 0)
    mask = cv2.erode(mask, None, iterations=5)
    mask = cv2.dilate(mask, None, iterations=5)

    # Encuentra contornos en la máscara
    contours, _ = cv2.findContours(
        mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        busqueda_iniciada=False
        largest_contour = max(contours, key=cv2.contourArea)
        area = cv2.contourArea(largest_contour)
        contador += 1
        area_prom = (area_prom + area) / contador

        #print("Área del contorno más grande:", area)

        if area >= area_prom:
            #print("area > 500")
            # Calcula el centro del contorno
            M = cv2.moments(largest_contour)
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])

            # Dibuja el contorno y muestra el color detectado
            cv2.drawContours(frame, [largest_contour], -1, (0, 255, 0),
3)
            cv2.putText(frame, color_actual, (cx - 20, cy - 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255),
2)

            # Verifica si el objeto está centrado en la pantalla
            if abs(cx - frame.shape[1] // 2) < 50:
```

```
            forward(0.02)
            #print("adel")
            #comienzo=False

        elif cx < frame.shape[1] // 2:
            turn_left(0.02)
            #print("izq")
        else:
            turn_right(0.02)
            #print("der")

        total_area = frame.shape[0] * frame.shape[1]

        if(cont == 5):
            color_actual = "Azul"
            area_g = 0.4

        if math.trunc(area) >= math.trunc(total_area) * area_g:
            # print("area mayor a 40%")

            if(cont == 5):
                balneario_found = True
                vuelta_entera()
                print("balneario")
                break

            imagen_encontrada = definirSignal(frame)
            senal_confirmada = afirmarSignal(imagen_encontrada)
            if (senal_confirmada):
                if(imagen_encontrada == 0):
                    pare_found = True
                    accion_pare()
                    print("pare")
                    cont += 1
                    continue
                elif(imagen_encontrada == 1):
                    adelante_found = True
                    accion_adelante()
                    print("adelante")
                    cont += 1
                    continue
                elif(imagen_encontrada == 2):
                    izquierda_found = True
```

```
                        giro_90_izq()
                        forward(1)
                        print("izquierda")
                        cont += 1
                        continue
                    elif(imagen_encontrada == 3):
                        derecha_found = True
                        giro_90_der()
                        print("derecha")
                        forward(1)
                        cont += 1
                        continue
                    elif(imagen_encontrada == 4):
                        prohibido_u_found = True
                        accion_giro_u()
                        color_actual = "Azul"
                        print("giro u")
                        cont += 1
                        continue
                else:
                    reverse(0.2)
    else:
        if not busqueda_iniciada:
            reverse(0.3)
            busqueda_iniciada = True
        turn_right(0.05)
        print("buscando meta")

    cv2.imshow("Video", frame)
    empezar = False
    k = cv2.waitKey(1)
    if k == 113:
        break

cap.release()
cv2.destroyAllWindows()
liberar_recursos(False)
```