import cv2 # Importa la biblioteca OPENCV para procesar imagenes

import pandas as pd # Importa la biblioteca PANDAS para crear el archivo .xlsx

import RPi.GPIO as GPIO # Importa la biblioteca GPIO para manejo de los pines de la Raspberry PI

import time # Importa la biblioteca tiempo para la obtencion de fecha y hora

import numpy # Importa la biblioteca numpy para operaciones numericas extensas

from picamera2 import Picamera2, Preview # Importa la biblioteca # Importa el modulo de la biblioteca para comunicar con el modulo de camara

import motor\_control as mc

import Cudaros as cdds # Modulo que contiene la funcion find\_squares

# ParÃ¡metro de velocidad general (ajustado entre 0 y 100)

velocidad\_general = 20 # Controla la velocidad general del carro

# Umbral para detectar el cuadrado de las seÃ±ales y empezar a avanzar

umbral = 160000

anc\_cam = 640

alt\_cam = 480

# ConfiguraciÃ³n para guardar datos en un archivo de Excel con los parametros de AcciÃ³n tomada, Momento de la acciÃ³n, Captura, Puntos por acciÃ³n

df = pd.DataFrame(columns=['AcciÃ³n', 'Timestamp', 'Imagen', 'Puntos'])

# Inicializar Variable puntos en 0 como Metrica de Rendimiento

puntos = 0

# Configuracion de porcentajes de area

MIN\_AREA\_PERCENT = 0.01 # Porcentaje minimo del area total (por ejemplo, 1%)

MAX\_AREA\_PERCENT = 0.10 # Porcentaje maximo del area total (por ejemplo, 10%)

# Iniciar un hilo para manejar las ventanas de OpenCV

cv2.startWindowThread()

# Configurar la camara Raspberry Pi

picam2 = Picamera2()

picam2.configure(picam2.create\_preview\_configuration(main={"format": 'XRGB8888', "size": (640, 480)}))

picam2.start()

frame = picam2.capture\_array() # Captura del frame representando los pixeles de la imagen

frame\_height, frame\_width, \_ = frame.shape

total\_area = frame\_height \* frame\_width

min\_area = MIN\_AREA\_PERCENT \* total\_area

max\_area = MAX\_AREA\_PERCENT \* total\_area

# FunciÃ³n para obtener el centroide de la lÃ­nea

def obtener\_centro(ancho\_cam, alto\_cam, umbral\_area=0.9):

frame = picam2.capture\_array() # Captura del frame representando los pixeles de la imagen

squares = cdds.find\_squares(frame, min\_area, max\_area) # Obtener cuadrados detectados

for square in squares:

# Calcula el momento para hallar el centroide

M = cv2.moments(square)

if M["m00"] != 0:

cx = int(M["m10"] / M["m00"])

cy = int(M["m01"] / M["m00"])

else:

cx, cy = 0, 0

# Calcula el centro de la cÃ¡mara

camera\_center\_x = ancho\_cam // 2

camera\_center\_y = alto\_cam // 2

# Calcula el error entre el centro del cuadrado y el centro de la cÃ¡mara

error\_x = camera\_center\_x - cx

error\_y = camera\_center\_y - cy

# Verifica si el cuadrado ocupa el Ã¡rea deseada

camera\_area = ancho\_cam \* alto\_cam

#should\_continue = max\_area / camera\_area < umbral\_area

should\_continue = max\_area < umbral\_area

return (error\_x, error\_y), should\_continue

else:

# No se detectÃ³ ningÃºn cuadrado

return None, True

# FunciÃ³n para capturar y guardar imagen con la acciÃ³n tomada

def registrar\_accion(accion, frame, puntos):

timestamp = time.strftime("%Y%m%d-%H%M%S") # Registro del momento exacto de la decisiÃ³n

file\_name = f'captura\_{timestamp}.jpg' # Nombre de la captura correspondiente al momento

cv2.imwrite(file\_name, frame) # Toma de captura

df.loc[len(df)] = [accion, timestamp, file\_name, puntos]

# Registro de la acciÃ³n, momento de captura, nombre de la captura, puntuaciÃ³n actual.

# FunciÃ³n para seguir la lÃ­nea

def iniciar\_recorrido():

mc.inicializar\_componentes()

error\_anterior = 0 # Inicializar valores en 0

global puntos # Variable global para Metrica de rendimiento

# Bucle para capturar y procesar cuadros en tiempo real

while True:

# Capturar un cuadro desde la camara

frame = picam2.capture\_array()

# Calcular el area total de la imagen

frame\_height, frame\_width, \_ = frame.shape

total\_area = frame\_height \* frame\_width

# Convertir los porcentajes en valores absolutos de area

min\_area = MIN\_AREA\_PERCENT \* total\_area

max\_area = MAX\_AREA\_PERCENT \* total\_area

# Detectar y dibujar cuadrados en el cuadro actual

processed\_frame = cdds.find\_squares(frame, min\_area, max\_area)

# Mostrar el cuadro procesado en una ventana

cv2.imshow("Detected Squares", processed\_frame)

# Inicio del bucle principal

(error, should\_continue) = obtener\_centro(anc\_cam, alt\_cam, umbral)

if error is not None:

error\_x, error\_y = error

if should\_continue:

print(f"Avanzando. Error X: {error\_x}, Error Y: {error\_y}")

# AquÃ­ llamas a tus funciones de movimiento

mc.controlar\_motores\_traseros(velocidad\_general)

mc.controlar\_motores\_delanteros(velocidad\_general)

mc.controlar\_direccion(error\_x, error\_anterior)

mc.ajustar\_velocidades(error\_x, velocidad\_general)

error\_anterior = error\_x # Error anterior se le asigna el valor del error que ya fue utilizado para el control PD

if abs(error\_x) < 10: # Si el error es menor a 10 pixeles

puntos += 4 # Se suman 10 puntos a las metricas de rendimiento

registrar\_accion('Avanzar hacia adelante', frame, puntos) # Se registra la acciÃ³n al dataframe

elif error\_x > 10: # Si el error es mayor a 10 pixeles

puntos -= 2 # Se restan 2 puntos a la metrica de rendimiento

registrar\_accion('Corregir a la derecha', frame, puntos) # Se registra la acciÃ³n al dataframe

elif error\_x < -10: # Si el error es mayor a 10 pixeles en sentido contrario

puntos -= 2 # Se restan 2 puntos a la metrica de rendimiento

registrar\_accion('Corregir a la izquierda', frame, puntos) # Se registra la acciÃ³n al dataframe

else:

print("El cuadrado ocupa el Ã¡rea deseada. Deteniendo.")

mc.detener\_carro()

puntos += 4

registrar\_accion('Clasificar imagen', frame, puntos) # Se registra la acciÃ³n al dataframe

# Funcion para clasificar la imagen mostrada

#respuesta = clasificacionRedNeuronal(binarizada)

#Accion(respuesta)

print('Clasificando seÃ±al')

else:

print("No se detectÃ³ ningÃºn cuadrado.")

puntos -= 5 # Se restan 5 puntos a la Metrica de rendimiento

registrar\_accion('Sin seÃ±al', frame, puntos) # Se registra la acciÃ³n al dataframe

mc.stop\_servo\_cam()

print("Vista binarizada")

#cv2.imshow("Vista", binarizada) # Crea una ventana de la imagen binarizada

# Salir del bucle al presionar la tecla 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows() # Destruye las ventanas creadas

mc.limpiar\_gpio()

df.to\_excel('registro\_acciones.xlsx', index=False) # Guarda el dataframe en un archivo .xlsx

# Llamada de la funciÃ³n para comenzar el recorrido

iniciar\_recorrido()

# Liberar los recursos y cerrar las ventanas

cv2.destroyAllWindows()