**Código Principal.py**

El siguiente código corresponde al script Detectron del programa para detectar y seguir a los deportistas en el test de Course – Navette.

**Librerías**

**1.** Funciones que se llaman desde otros códigos y que se utilizan en el código principal.

from video import FileVideoStream

from Detectron import deteccion\_seguimiento

from aux\_functions import \*

La primera librería importa la clase que permite hacer la lectura del video en hilos, la segunda librería importa la clase que permite realizar la detención y el seguimiento a los deportistas basado en el centroide de los recuadros que identifican a cada uno, la tercera librería importa la clase que permite realizar la obtención del cambio de perspectiva o vista de pájaro. Cada una de estas librerías son explicadas a detalle más adelante.

**2.** Librerías de paquetes de funciones necesarias.

import numpy as np

import cv2

import math

import time

*Numpy*: Numpy es una librería de Python que permite trabajar de forma sencilla con vectores y matrices de gran tamaño y multidimensionales. Numpy permite declarar arrays de distintas dimensiones capaces de almacenar una gran cantidad de datos del mismo tipo relacionados entre sí; de igual forma posee una gran cantidad de métodos para manipular dichos arrays, acceder a su información y realizar el debido procesamiento. La documentación oficial de esta librería se puede encontrar en [1].

*cv2*: cv2 es el paquete a través del cual se importan las diversas funciones de OpenCV (Open Source Computer Vision), esta última, es la librería más grande que existe actualmente, de visión por computador en términos de funciones que contiene. Actualmente, OpenCV contiene implementaciones de más de 2500 algoritmos y está disponible para múltiples lenguajes de programación tales como Python, Java y C ++. La documentación oficial de esta librería se puede encontrar en [2].

*math*: El módulo math de Python proporciona acceso a las funciones definidas por el estándar C. Todos los métodos de estas funciones se utilizan para números enteros reales, y no para números complejos. La documentación oficial de esta librería se puede encontrar en [3].

*time*: El módulo time de Python proporciona un conjunto de funciones para trabajar con fechas y/o horas. La documentación oficial de esta librería se puede encontrar en [4]

**Datos de Entrada por medio de consola.**

Mediante las siguientes líneas de código se declaran los datos que se van a ingresar al programa por medio de la consola.

video = input("Ingrese la ruta del video a analizar: ")'

print("")

salida = input("Ingrese la ruta de salida deseada para el video procesdo: ")

print("")

seg\_inicio = int(input("Ingrese el tiempo de inicio del video en segundos: "))

print("")

El primer input corresponde a la ruta del video a analizar, el segundo input corresponde a la ruta de salida del video ya procesado y por último, tenemos un input el cual corresponde a los segundos en que inicia el test de Course Navette.

**Creación de archivos de acuerdo con el número de jugadores.**

Mediante el siguiente for se realizará la creación de las listas, que contienen los estados de aprobación de cada jugador y los archivos con los respectivos identificadores, donde será almaceno los datos evaluados.

for i in range(n\_corredores) :

list\_estados.append(list(repeat(0, 142)))

filename = 'Corredor\_' + str(i)+'.txt'

f= open(filename,"w+")

tex = 'Registro de evaluaciones para el corredor ID: '+ str(i)+'\n'

f.write(tex+'\n')

f.close()

Como primera medida se crea las listas para cada identificador dependiendo el numero de corredores, seguido a esto se crean los archivos “.txt” en los cuales se ira registrando, la etapa, el tramo, el estado aprobación o desaprobación y por último su capacidad pulmonar.

**Lectura de archivo con los puntos de calibración**

El siguiente if es encardo de leer el archivo, que contiene los puntos que conforman el escenario deportivo del test.

if not os.path.isfile('puntos.data'):

print("[INFO] Seleccion no realizada!")

print("")

print("La selección de los puntos de calibración no ha sido realizada, para iniciar el analisis del video debe ejecutar primero el archivo Seleccion\_ROI.py")

sys.exit()

else:

with open('puntos.data', 'rb') as filehandle:

puntos = pickle.load(filehandle)

four\_points = puntos[0:4]

La condición evalúa, si existe el archivo o no, dado el caso que este no exista generara un letrero informando que la selección de los puntos no fue realizada y posteriormente el paso a realizar. Posteriormente si existe el arco, se procede a realizar la lectura de este y a guardar los datos que vienen, es decir la selección de los puntos.

**Función de análisis para las posiciones de los jugadores**

La siguiente función es la encarga de evaluar si los deportistas aprueban o no aprueban el tramo, a su ves también es la encargada de determinar cuando el jugador es descalificado, al no cumplir con los requerimientos del test de Course Navette y por ultimo se encargada de guardar en los archivos “.txt” los datos de evaluación correspondientes a cada uno de los deportistas. Esta función también es la encargada, de realizar lo cálculos para la capacidad pulmonar, esto dependiendo para cada uno de los casos.

**1.** Ciclo for para iterar sobre los objetos detectados en el escenario deportivo.

Para obtener las condenas de los jugadores que están realizando el análisis, se itera sobre la lista de los puntos detectados en el escenario deportivo.

Antes de definir el ciclo for, se definen unas variables globales las cuales van a contener información clave para poder realizar los cálculos para la ejecion del test. En la definición del ciclo for se utiliza la función enumerate, para determinar el rango en el que se va a iterar y la extracción de la coordenada a analizar.

**2.** Ciclo if para la clasificación de los putos aprobados.

Para realizar el análisis de los jugadores que aprobaran el tramo, se implementó unas coordenadas, las cuales están impuestas por las dimensiones del escenario deportivo. En caso de que los jugadores, estén dentro del rango, se emitirá una alerta de aprobación y se procede a realizar el calculo de la capacidad pulmonar.

En caso de alguna coordenada no cumpla, se implementa un ciclo if el cual evaluara los casos en que el deportista no apruebe el test.

def analisis\_posiciones\_1(lista\_corredores,Lista\_ides,n\_etapa,n\_tramo):

global tramo\_analizado, n\_corredores, Velocidades

for i,coord in enumerate(lista\_corredores):

coord\_x, coord\_y = coord # coordena en y de punto

if (200 >= coord\_y > 0 ) or (1480 >= coord\_y >= 1280):

print("Corredor ID: ",Lista\_ides[i],"APROBADO")

VO\_MAX = round(((5.857\*Velocidades[int(n\_etapa)-1])-19.458),2)

capaciPul= "su capacidad pulmunar es de: "+ str(VO\_MAX)+' VO2max \n'

registrar ='Etapa ' + n\_etapa +' - ' + ' Tramo ' + str(n\_tramo) + ' El corredor ' + ' aprobo y '+ capaciPul

else:

if coord\_y >0 and coord\_y <=1480 and i <= n\_corredores-1:

print("Corredor ID: ",Lista\_ides[i],"RECHAZADO")

list\_estados[Lista\_ides[i]][tramo\_analizado] = 1

tramos\_rechazados = sum(list\_estados[Lista\_ides[i]])

registrar ='Etapa '+ n\_etapa +' - '+ ' Tramo ' + str(n\_tramo) + ' El corredor no aprobo\n'

if tramos\_rechazados >=2:

print('El corredor ha sido decalificado: ',i)

Descali ='-------------------El corredor fue descalificado en la etapa '+ n\_etapa + ' y tramo ' + str(n\_tramo) + '-----------------\n'

VO\_MAX = round(((5.857\*Velocidades[int(n\_etapa)-1])-19.458),2)

filename = 'Corredor\_'+str(Lista\_ides[i])+'.txt'

f= open(filename,"a")

capaciPul= "La capacidad pulmunar es de: "+ str(VO\_MAX)+' VO2max \n'

f.write(Descali+'\n')

f.write(capaciPul+'\n')

f.close()

if i <= n\_corredores-1 :

filename = 'Corredor\_'+str(Lista\_ides[i])+'.txt'

filename1 = 'corredor\_'+str(i)+'.txt'

f= open(filename,"a")

f.write(registrar+'\n')

f.close()

tramo\_analizado = tramo\_analizado + 1

Este ciclo tendrá como parámetros de evaluación los rangos, en los que no se cumple el test, dado el caso de que un jugador no cumpla, se generara un alerta en el que se dira que el corredor con el identificador fue rechazado, posteriormente se evalúa mediante un ciclo if si el deportista ya ha sido penalizado dos veces, si se da el caso este procede, emitiendo una alerta de descalificación y la capacidad pulmonar alcanzada por el deportista.

**3.** Ciclo if encargado de almacenar en los datos en archivos “.txt”

Por último tenemos un if, el cual almacena los datos para cada jugador, esto dependiendo su identificador para su respectiva clasificación en los archivos.

**Variables y Objetos a Inicializar.**

**1.** Lectura de frames del video y objeto para guardar el video.

print ('[INFO] starting video file thread...')

fvs = FileVideoStream(args['video']).start()

time.sleep(1.0)

writer = None

A través de la función FileVideoStream, importada del código imutilsvideo.py, se inicia la lectura de los frames del video a analizar, el único parámetro de entrada de esta función es la ruta del video, la cual se llama a través del método args del argparse. El método start() de la función FileVideoStream hace que el buffer empiece a almacenar los frames que se van leyendo. A partir de esta línea, y en el resto del código, esta función será invocada como fvs.

A través del método sleep() de la librería time se agrega al procesamiento un delay de 1 segundo, el cual se utiliza para permitir que el buffer empiece a almacenar datos antes de entrar al ciclo while.

La última línea de código inicializa a None el writer que va a guardar el video final producto del procesamiento.

**2.** Frames por Segundo del video y contadores.

A través del método fps() de la función FileVideoStream, previamente importada, se obtienen los frames por segundo del video a analizar; esta cifra es redondeada, en la segunda línea de código, a tres cifras significativas.

framepersec = fvs.fps()

frames=round(framepersec,3)

minu=0;con=0;cont=0;tiempo=[]

En la última línea de código se inicializan a cero una serie de variables, contadores y listas que serán utilizadas dentro del ciclo while para llevar el conteo de la duración del video analizado. La variable minu se utiliza para indicar los minutos, el contador con es el encargado de llevar la cuenta de los minutos, mientras que cont lleva la cuenta de los frames que se han procesado, la lista de tiempo se utiliza para almacenar el valor de frames procesados.

**Ciclo While**

Luego de haber importado e inicializado las variables y funciones necesarias para el procesamiento del programa, se crea un ciclo while, el cual va a iterar sobre el buffer de frames que se han leído del video, para esto, hace uso del método more() de la función FileVideoStream; este método crea un valor booleano que permanece verdadero mientras existan frames almacenados en el buffer sobre los cuales se pueda iterar, al hacer uso de esta condición en el ciclo while se asegura de que el ciclo se va a seguir repitiendo mientras hayan frames por procesar, en el momento en que el buffer se quede sin frames, el valor booleano cambia a falso y el ciclo while se rompe.

**1**. Lectura de frames.

Lo primero que se realiza dentro del ciclo while es la lectura de los frames y su almacenamiento en la respectiva variable, la lectura se realiza a través del método read() de la función fvs. Seguido a esto, se utiliza un ciclo if para el determinado caso de que la variable frame se encuentre vacía, de ser así el ciclo se rompe.

while fvs.more():

frame = fvs.read()

if frame is None:

break

**2.** Escritura del archivo de salida.

Para grabar el video final producto del procesamiento del programa se define el objeto a utilizar.

if salida is not None and writer is None:

fourcc = cv2.VideoWriter\_fourcc(\*'MJPG')

writer = cv2.VideoWriter(args['output'], fourcc, framepersec,

(frame.shape[1], frame.shape[0]), True)

A través del anterior ciclo if el programa revisa si ya existe un archivo de salida en la ruta suministrada (salida) y si el objeto que va a guardar el video no se ha definido, de ser así, el programa procede a crear dicho objeto a través de la función cv2.VideoWriter [17], el cual tiene los siguientes parámetros de entrada:

* Ruta del archivo de salida.
* El código del CODEC usado para comprimir los frames. Una lista de los códigos disponibles se puede conseguir en el link [18]. Para establecer uno de estos códigos, se hace uso de la función cv2.VideoWriter\_fourcc.
* La tasa de frames por segundo a la que se va a guardar el video. Para este caso se utiliza la misma tasa del video original (framepersec).
* El tamaño de los frames del video. frame.shape[1] corresponde al alto y frame.shape[0] al ancho.
* isColor: Esta es una bandera que se utiliza para indicarle al codec, en caso de que sea verdadera, que los frames del video son a color y así se deben guardar.

**3.** Transformación de perspectiva.

Para obtener la vista de pájaro, es necesario llamar la función, que se encuentra alojada en el código aux\_fuctions.py, la cual es llamada mediante get\_camera\_perspective(), esta nos permite retornar una matriz, la cual es utilizada para realizar el cambio de perspectiva, esta matriz de transformación, necesita 4 puntos en cv2.polylines(frame\_copy, [pts], True, (0, 255, 255), thickness, cv2.LINE\_AA)la imagen de entrada y los puntos correspondientes en la imagen de salida.

M, Minv = get\_camera\_perspective(frame, puntos[0:4])

pts = np.array([four\_points[0], four\_points[1], four\_points[3], four\_points[2]], np.int32)

thickness=3

warped\_image = cv2.warpPerspective(frame, M, (IMAGE\_W, IMAGE\_H))

warped\_image = cv2.resize(warped\_image,None,fx=0.5, fy=0.5,

interpolation = cv2.INTER\_LINEAR)

cv2.polylines(frame\_copy, [pts], True, (0, 255, 255), thickness, cv2.LINE\_AA)

Finalmente aplicamos la función cv2.warpPerspective() cuyos argumentos son: imagen de entrada, Matriz de transformación de 3x3 y tamaño de la imagen de salida (columnas, filas). Posteriormente realizamos la representación del escenario deportivo en el video de salida.

**4.** Detención de personas y creación de imagen para el escenario deportivo.

Para la detención de personas utilizamos la función deteccion\_seguimiento() alojado en condigo de Detecton.py, el cual, nos retorna la imagen con el dibujo de rectángulos delimitadores y centroides, al igual que listas en python [(),()...,()] con los centroides asociados y los identificadores de cada uno de ellos. Estos puntos serán mapeados a la imagen que representara el escenario deportivo.

Posteriormente se crea una imagen de ceros. En la cual serán dibujados los centroides de los jugadores, a su vez se crean las líneas de meta con el comando cv2.line(), esto con el fin de realizar la representación grafica del escenario deportivo.

frame, centros, ID\_cen = deteccion\_seguimiento(frame\_copy\_1)

bird\_image\_1 = np.zeros((int(frame\_h \* scale\_h)+400, int(frame\_w \* scale\_w), 3), np.uint8)

bird\_image\_1[:] = (41, 41, 41)

line\_thickness = 2

cv2.line(bird\_image\_1, (0, 200), (int(frame\_w \* scale\_w), 200), (0, 0, 255), thickness=line\_thickness)

cv2.line(bird\_image\_1, (0, int(frame\_h \* scale\_h)+200), (int(frame\_w \* scale\_w), int(frame\_h \* scale\_h)+200), (0, 0, 255), thickness=line\_thickness)

**5.** Identificación de centroides en la transformación de perspectiva.

En el siguiente ciclo identificaremos los centroides, que están en todo el frame y posteriormente se realiza el cambio de perspectiva para cada uno de estos y así poderlos representar en la representación del escenario deportivo.

if len(centros)>0:

node\_radius = 10

color\_node = (192, 133, 156)

thickness\_node = 20

pts\_c = np.float32([centros[:]])

warped\_pt = cv2.perspectiveTransform(pts\_c, M)[0]

lista\_corredores = []

Lista\_ides = []

for i in range (len(warped\_pt)):

warped\_pt\_scaled\_1 = (int(warped\_pt[i][0] \* scale\_w), int(warped\_pt[i][1] \* scale\_h)+200)

cv2.circle(bird\_image\_1, warped\_pt\_scaled\_1, 4, (0,255,0), 3, cv2.LINE\_AA)

lista\_corredores.append(warped\_pt\_scaled\_1)

Lista\_ides.append(ID\_cen[i])

para realizar la transformación de perspectiva como se venia explicando en la sección 3, procedemos a utilizar la función cv2.warpPerspective() cuyos argumentos son: imagen de entrada, Matriz de transformación de 3x3 y tamaño de la imagen de salida (columnas, filas). Permitiendo tener el respectivo cambio de los puntos para realizar su respectivo análisis. Seguido a esto se realiza un ciclo for en el cual se recorren la cantidad de puntos transformados, dentro de este vamos a dibujar los puntos mediante la función de cv2.Circle(), la cual va a permitir la representación del escenario deportivo, por ultimo se procede a guardar los puntos, en una lista, la cual será utilizada para realizar la evaluación del test de Course Navette.

**6.** Tiempo de duración del video.

Para poder llevar la cuenta de la duración del video durante el procesamiento se hacen uso de las siguientes líneas de código.

seg=con/frames

tiempo.append(cont)

if seg >= 60:

con=0

minu +=1

time = f"{minu:02d}:{int(seg):02d}"

print(time)

text2 = 'Tiempo: '+ time

cv2.putText(

frame,

text2,

(20,30),

cv2.FONT\_HERSHEY\_COMPLEX,

1.2,

(0, 0, 255),

2,

lineType=cv2.LINE\_AA,

)

En la primera línea para obtener el número de segundos se divide el número de frames procesados entre la tasa de frames por segundo del video; seguidamente, se almacena el valor de frames procesados en la variable tiempo, previamente inicializada.

Para obtener el número de minutos se crea un ciclo if, el cual actualiza el valor del contador cont y aumenta en uno el valor de la variable minu cada vez que el valor de los segundos es mayor o igual a sesenta.

Para obtener el valor del tiempo en el formato minutos:segundos se utiliza la notación que aparece en la sexta línea de código; finalmente, esta variable es imprimida en el video de salida a través de la función cv2.putText.

**7.** Análisis de Etapas

Este if se encarga de iniciar la etapa, finalizarla y posteriormente activar la etapa siguiente, cabe resaltar que la estructura de este if es la misma para cada etapa, solo varia en valores numérico, para cada etapa.

if etapa\_2 == True and seg\_etapas > 60.0:

seg\_etapa = seg\_etapas - 60.0

seg\_etapa = float(np.round(seg\_etapa, 2))

if (tramo\*8.00-0.03) <= seg\_etapa <= (tramo\*8.00+0.03):

frame[1000:1080,0:1920] = [0,255,0]

analisis\_posiciones\_1(lista\_corredores,Lista\_ides,'2',tramo)

print("Analizando Posiciones")

tramo = tramo + 1

else:

frame[1000:1080,0:1920] = [255,255,255]

if tramo == 8: # se da paso a etapa\_3

tramo = 1

etapa\_2 = False

etapa\_3 = True

cv2.putText(frame,'Etapa 2', (850,1050), cv2.FONT\_HERSHEY\_COMPLEX,

1.1,(0, 0, 255), 2, lineType=cv2.LINE\_AA,)

texto = 'Tramo ' + str(tramo)+ ' ' + 'segundo\_ ' + str(seg\_etapa)

cv2.putText(frame,texto, (1020,1050),cv2.FONT\_HERSHEY\_COMPLEX,

1.1,(0, 0, 255), 2, lineType=cv2.LINE\_AA,)

La estructura de este código esta compuesta por un if principal, básicamente evalúa si la etapa fue activada y el que inicio en segundos de la etapa sea mayor al de la etapa. Siguiendo con la estructura, tenemos un if definido por un rango, que evaluara los segundos en el que ocurre el «beep», dando paso a la función que es la encargada de determinar el estado de aprobación o desaprobación del jugador, así como también su capacidad pulmonar, por último, tenemos un if el cual, cerrara la etapa en desarrollo, abrirá la siguiente etapa siempre y cuando se cumpla el numero de tramos para la etapa en curso. Finalizando tendremos las siguiente funcione, cv2.putText() la cual era la encargada de dibujar en el frame, la etapa, el tramo y el tiempo e segundos de la etapa que está en desarrollo.

**8.** Guardar los frames procesados.

cv2.imwrite('Frame.png', frame)

if writer is not None:

writer.write(frame)

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

Luego de realizar todo el procesamiento descrito anteriormente, se procede a guardar los frames del video ya analizados, para esto se hace uso de la función cv2.imwrite [23], la cual guarda la imagen con el nombre asignado.

Seguido a esto, se verifica si el objeto encargado de guardar el video final se encuentra o no vacio, y de no estarlo se procede a guardar los frames a través del método write.

A través de la función cv2.waitKey [24] se le da al programa un método de escape en caso de necesitarlo, es decir, se escoge una letra, en este caso ‘q’, la cual de ser presionada, rompe el ciclo cerrando la ventana del procesamiento.

**Limpieza del Programa**

Luego de que el ciclo while se deje de ejecutar, es decir, cuando el procesamiento del video haya terminado, se utilizan las siguientes líneas para realizar una limpieza del programa.

if writer is not None:

writer.release()

#Limpieza

cv2.destroyAllWindows()

fvs.stop()

A través del ciclo if se comprueba si el objeto encargado de guardar el video ya ha terminado de almacenar los frames procesados, de ser así, se libera a través del método reléase().

Con la función cv2.destroyAllWindows [25] se destruyen todas las ventanas abiertas durante el procesamiento del programa, y finalmente, mediante el método stop de la función fvs se detiene la lectura de frames del video de entrada.

**Código Detectron.py**

El siguiente código corresponde al script Detectron del programa para detectar y seguir a los deportistas en el test de Course – Navette.

**Librerías**

**1.** Funciones que se llaman desde otros códigos y que se utilizan en el código principal.

from centroidtracker import CentroidTracker

La librería importa la clase que permite realizar el seguimiento a los deportistas basado en el centroide de los recuadros que identifican a cada uno. Cada una de estas librerías es explicada a detalle más adelante.

**2.** Librerías de paquetes de funciones necesarias.

import numpy as np

import cv2

import math

import time

**3.** Librerías del Paquete de Detectron2.

import detectron2

from detectron2 import model\_zoo

from detectron2.utils.logger import setup\_logger

setup\_logger()

from detectron2.engine import DefaultPredictor

from detectron2.config import get\_cfg

from detectron2.utils.visualizer import Visualizer

from detectron2.data import MetadataCatalog

from detectron2.structures import Boxes, BoxMode, pairwise\_iou

La primera librería importa todas las funciones y métodos asociados al software de detectron2, de estas funciones las que se utilizan para la implementación del programa son las siguientes:

*model\_zoo*: Es un archivo que contiene la colección de modelos pre-entrenados de Detectron2. Estos modelos se pueden encontrar en [5].

*detectron2.utils.logger. setup\_logger*: Inicializa el logger o protocolo de registro de eventos de Detectron2 y establece su nivel de registro de información a “DEBUG”, lo que significa que se ofrece al usuario información mucho más completa, tal como los valores de las variables depuradas. La información completa de este módulo se encuentra en [6].

*detectron2.engine. DefaultPredictor*: Crea un predictor simple con la configuración dada que se ejecuta en un solo dispositivo para una sola imagen de entrada. La información completa de este módulo se encuentra en [7].

*detectron2.config.get\_cfg*: Obtiene una copia de la configuración por defecto de Detectron2. La información completa de este módulo se encuentra en [8].

*detectron2.utils.visualizer.Visualizer*: La clase de Visualizer se encarga de dibujar la información sobre la detección/segmentación de imágenes. La información completa de este módulo se encuentra en [9].

*detectron2.data.MetadataCatalog*: El MetadataCatalog es un diccionario global que provee acceso a la metada de un determinado dataset. La información completa de este módulo se encuentra en [10].

*detectron2.structures.Boxes*: Esta estructura almacena una lista de recuadros de tipo torch.Tensor Nx4. Soporta algunos métodos comunes relacionados con los rectángulos y se comporta como un Tensor. La información completa de este módulo se encuentra en [11].

*detectron2.structures.BoxMode*: Esta clase enumera las distintas formas que tiene Detectron2 de representar un recuadro. La información completa de este módulo se encuentra en [12].

*detectron2.structures.pairwise\_iou*: Dadas dos listas de recuadros de tamaños N y M, esta clase se encarga de computar la intersección sobre unión (IoU) entre todos los NxM pares de recuadros. La información completa de este módulo se encuentra en [13].

**Variables y Objetos a Inicializar.**

**1.** Configuración de Detectron2 y CentroidTracker.

cfg = get\_cfg().

cfg.merge\_from\_file(model\_zoo.get\_config\_file('COCO-Keypoints/keypoint\_rcnn\_R\_50\_FPN\_3x.yaml'

))

cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.7

cfg.MODEL.WEIGHTS = \

model\_zoo.get\_checkpoint\_url('COCO-Keypoints/keypoint\_rcnn\_R\_50\_FPN\_3x.yaml'

)

predictor = DefaultPredictor(cfg)

ct = CentroidTracker()

A través de las anteriores líneas de código se establece la configuración básica a utilizar de Detectron2 en el programa y se inicializa el centroidtracker. En la primera línea, el comando get\_cfg() se utiliza para obtener la configuración por defecto del detector, la cual se asigna a la variable cfg.

En la segunda línea, la función model\_zoo.get\_config\_file devuelve la ruta al archivo de configuración incorporado en Detectron2 [14] mientras que cfg.merge\_from\_file carga los valores del archivo devuelto por el anterior método. En la tercera línea, el método MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST permite establecer el valor de threshold a utilizar para el modelo. En la cuarta línea, la función model\_zoo.get\_checkpoint\_url devuelve la dirección URL del modelo entrenado usando la configuración dada [15].

Finalmente, a través de la clase DefaultPredictor () se crea un predictor simple con la configuración dada (cfg), el cual se ejecuta en un solo dispositivo para una sola imagen de entrada. La información completa de esta clase se encuentra en [16].

En la última línea de código se llama al CentroidTracker, a través de la función CentroidTracker(), la cual se importa del código centroidtracker.py. A partir de esta línea, y en el resto del código, esta función será invocada como ct.

**Clase de Detección de Seguimiento**

Luego de haber importado e inicializado las variables y funciones necesarias para el procesamiento del programa, se crea una clase, el cual va a iterar sobre el buffer de frames que se ingresa en la clase.

**1.** Conversión del espacio de color.

frame = cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR)

Debido a que Detectron2 solo acepta imágenes en formato BGR, se realiza la conversión del espacio de color RGB a BGR a través de la función de cv2, cvtColor.

**2.** Objeto del predictor.

Detectron2 realiza las detecciones en base a las imágenes o frames que se leen del video a analizar, en las siguientes líneas de código, se configura la forma en que se pasan dichas imágenes al predictor.

outputs = predictor(frame)

instances = outputs['instances']

scores = instances.scores.cpu().numpy()

print ('Numero de Identificados: ', len(instances))

En la primera línea se llama al objeto del predictor, creado por fuera del ciclo while, y se le pasa como argumento el frame actual, de esta forma, el programa hace la detección de las personas sobre esa imagen de entrada. Los resultados producto de la detección se almacenan en la variable outputs en forma de diccionario.

En la segunda línea, se accede a una instancia del diccionario (‘instances’), la cual almacena los campos de los recuadros, los keypoints y los confidence scores, entre otros. Este último parámetro, como se explicó anteriormente, se utiliza para filtrar detecciones incorrectas del programa, y es el que se obtiene en la tercera línea de código mediante el instances.scores. Debido a que el tipo de dato de este y los demás parámetros es un tensor, es necesario realizar la conversión a un array de numpy, para su posterior manipulación; esta operación se realiza primero copiando el tensor a la cpu (.cpu()) y luego convirtiéndolo al array (.numpy()).

Al imprimir la longitud de instances (len(instances)) se está imprimiendo el número de objetos detectados por el programa.

**3.** Ciclo for para iterar sobre los objetos detectados.

Para obtener las coordenadas de los recuadros y los keypoints, se crea un ciclo for para iterar sobre los objetos detectados por Detectron2.

Antes de definir el ciclo for, se inicializa la lista que va a almacenar las coordenadas de los recuadros de los objetos detectados (rects) y el contador que se va a utilizar para iterar sobre dichos objetos.

En la definición del ciclo for se utiliza la función range [19] para determinar el rango en el que se va a iterar; esta función tiene dos parámetros, el primero es el punto de inicio (0) y el segundo es el punto final (N° de objetos identificados), de no indicarlo el valor del step o paso se asume en 1.

rects = []

i = 0

for i in range(0, len(instances)):

confidence = scores[i]

if confidence > args["confidence"]:

#Boxes

boxes = instances.pred\_boxes.tensor.cpu().numpy()

#print('Coordenadas Recuadros')

#print(str(i)+'-',boxes[i])

#Keypoints

has\_keypoints = instances.has('pred\_keypoints')

if has\_keypoints:

keypoints = instances.pred\_keypoints.cpu().numpy()

keypoints = keypoints[:, :, :2]

# print('Keypoints identificados')

# print(str(i)+'-',keypoints[i])

#Tracking

box = boxes[i]

rects.append(box.astype('int'))

(startX, startY, endX, endY) = box.astype('int')

cv2.rectangle(frame, (startX, startY), (endX, endY), (0,

255, 0), 2)

En el momento en que se entra al ciclo for, el programa crea una variable (confidence) para iterar sobre los valores de scores y de esta forma filtrar las detecciones erróneas; si el valor de dicho threshold es mayor al mínimo ingresado vía línea de comando args["confidence"] se obtienen las coordenadas de los recuadros y los keypoints para las detecciones filtradas. Para la obtención de estas coordenadas se realiza lo siguiente:

* Boxes: Para obtener las coordenadas de los recuadros se utiliza el objeto de instances y se accede a la clase pred\_boxes, que es la que almacena dichos datos. Al igual que para scores se debe realizar un proceso de conversión, sin embargo, ya que pred\_boxes es de tipo Torch, es necesario convertir el dato primero a un tensor.
* Keypoints: Para obtener las coordenadas de los keypoints, primero se hace la comprobación de la existencia de este dato en el objeto de instances a través de la clase has(‘pred\_keypoints’), si esa comprobación es verdadera, es decir, si existen datos para esa clase, se obtienen las coordenadas para los keypoints y se realiza la conversión de tensor a array de numpy. Debido a que el array que almacena los keypoints tiene tres datos, la coordenada en X, la coordenada en Y, y la visibilidad de cada punto, se accede solo a los primeros dos datos de interés a través de keypoints[:, :, :2].
* Tracking: Para comenzar con el proceso de tracking, se empieza por iterar sobre las coordenadas de los recuadros, el tipo de dato se cambia a int o entero a través de la función astype, y se almacenan estos datos en una nueva variable (recs), que será posteriormente pasada al seguidor. Para dibujar las coordenadas de los recuadros sobre el frame se utiliza la función cv2.rectangle [20]

**4.** Centroidtracker

Luego de cada iteración del ciclo for, la variable de recs se va llenando con las coordenadas de los recuadros de los objetos detectados por Detectron2, esta variable es la que utiliza el centroidtracker para hacer el seguimiento de dichos objetos; en el código actual se utiliza el método uptade de la función ct para actualizar el centroidtracker con las coordenadas que se obtienen frame a frame, todo el proceso que realiza el seguidor con estas coordenadas se explica de forma completa más adelante en el código centroidtracker.py

objects = ct.update(rects)

Idcen, Idrec = objects

#print("lista completa",objects)

Del objeto que se utiliza para obtener los resultados del seguidor (objects), se obtienen dos listas, la primera, Idcen, almacena los centroides calculados junto a su correspondiente Id o identificador, y la segunda, Idrec, almacena las coordenadas de los recuadros correspondientes a cada identificador.

**5.** Dibujo de centroides.

for (objectID, centroid) in Idcen.items():

coordenada = centroid

text = 'Id{}'.format(objectID)

cv2.putText(

frame,

text,

(centroid[0] - 20, centroid[1] - 20),

cv2.FONT\_HERSHEY\_COMPLEX,

0.8,

(0, 0, 0),

1,

lineType=cv2.LINE\_AA,

)

cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0),

-1)

Para dibujar los centroides calculados por el seguidor se crea un ciclo for que itera sobre los ítems de la lista de centroides, dichos ítems son el Id del objeto (objectID) y el centroide correspondiente (centroid). Para ubicar como texto el número del identificador sobre cada deportista se utiliza la función cv2.putText [21], esta acepta como parámetros de entrada la imagen donde se va a ubicar el texto, el texto que se va a dibujar, las coordenadas en donde se va a ubicar, que en este caso son las coordenadas del centroide que corresponda, el tipo de letra, el tamaño de la letra, el color de la letra (en formato BGR), el grosor de la línea de la letra y el tipo de línea.

Para dibujar el punto del centroide se utiliza la función cv2.circle [22], esta acepta como parámetros de entrada la imagen en donde se va a dibujar, las coordenadas del centro del circulo, el radio del circulo, el color del circulo y el grosor de la línea del circulo, para valores negativos en este último parámetro, la función asume que el circulo debe ser relleno.

**Código video.py**

El siguiente código se utiliza para leer los frames del video a procesar, a través de programación en hilos, y almacenarlos en un buffer o en un Queue [26].

**Librerías.**

**1.** Librerías de paquetes de funciones necesarias.

from threading import Thread

import sys

import cv2

import time

*threading*: El módulo threading construye interfaces de procesamiento en hilos de alto nivel [27]. De este módulo se importa la clase Thread [28], la cual, es usada para crear e iniciar hilos en el lenguaje de programación de Python.

*sys*: El módulo sys de Python [29] provee acceso a algunas variables utilizadas o mantenidas por el interprete y a funciones que interactúan fuertemente con el interprete.

**2.** Librería de la estructura Queue.

if sys.version\_info >= (3, 0):

from queue import Queue

else:

from Queue import Queue

Para importar la clase de Queue [30] del módulo queue [31] se implementa el ciclo if que aparece en las anteriores líneas de código, esto debido a que el nombre del módulo es diferente según la versión de Python que se esté utilizando; para conocer la versión de Python del sistema se utiliza la función versión\_info [32] del módulo sys.

**Constructor de la clase FileVideoStream**

class FileVideoStream:

def \_\_init\_\_(self, path, transform=None, queue\_size=128):

self.stream = cv2.VideoCapture(path)

self.stopped = False

self.transform = transform

self.Q = Queue(maxsize=queue\_size)

self.thread = Thread(target=self.update, args=())

self.thread.daemon = True

En las anteriores líneas de código se define el constructor para la clase FileVideoStream, la cual, toma como parámetros de entrada un argumento requerido seguido por uno opcional:

* path: La ruta del archivo de video a procesar. (Requerido)
* queueSize: Este parámetro representa el número máximo de frames a almacenar en el queue; por defecto, este valor se ha establecido en 128, pero puede ser modificado por el usuario dependiendo de las dimensiones de los frames del video a analizar y de la cantidad de memoria de la cual se pueda prescindir. (Opcional)

En la línea de self.stream se crea la instancia para el objeto de cv2.VideoCapture [33] al pasar la ruta del video. En la siguiente línea de código, se inicializa un valor booleano para indicar si el procesamiento en paralelo debe ser detenido junto a la estructura actual del Queue, inicializada en la línea de self.Q.

La instancia de self.transform se utiliza en caso de que se necesiten hacer transformaciones a nivel de los frames del video.

El constructor de la clase Thread, en la línea self.thread, se utiliza para crear el objeto del subproceso, de los parámetros que acepta, target es el objeto a invocar por el método run(), es decir, el proceso a ejecutar; por otro lado, args es la tupla de argumentos para invocar al target u objetivo, su valor por defecto es (). En [28] se encuentra información más detallada sobre los parámetros de este constructor.

La instancia de self.thread.daemon se utiliza para etiquetar al subproceso como “daemon thread”. El significado de esto reside en que cuando todo el programa en Python se cierra los daemon thread también terminan.

**1**. Método start.

def start(self):

self.thread.start()

return self

El método start da inicio a un hilo o subproceso separado del proceso inicial. Este hilo va a llamar al método .update que se define en el siguiente bloque.

**2**. Método update.

El método update es el responsable de leer y decodificar los frames del archivo de video, así como también, de mantener la estructura de datos del Queue actual.

def update(self):

while True:

if self.stopped:

break

if not self.Q.full():

(grabbed, frame) = self.stream.read()

if not grabbed:

cv2.waitKey()

break

if self.transform:

frame = self.transform(frame)

self.Q.put(frame)

else:

time.sleep(0.1)

self.stream.release()

La clave del procesamiento paralelo del código reside en este método, ya que, este bloque de código se ejecuta en un hilo aparte.

En el while True se empieza a iterar sobre los frames del archivo de video, si el indicador de stopped es establecido a verdadero, el hilo se rompe, de lo contrario, y si la estructura del queue no está llena, se lee el siguiente frame del video con self.stream.read(); seguido a esto se comprueba si ya se ha alcanzado el final del video con el ciclo if not grabbed y, dado el caso, se para el procesamiento, de igual forma se comprueba si se necesitan hacer transformaciones en el frame del video, y finalmente se actualiza la estructura del queue al utilizar el método put en la instancia self.Q. En el caso de que el queue este lleno, se da al procesamiento un descanso de 10ms y cuando el ciclo while se interrumpa se libera la ejecución del video.

**3.** Método fps.

def fps(self):

return self.stream.get(cv2.CAP\_PROP\_FPS)

El método fps se utiliza para obtener la tasa de frames por segundo del video que se esta utilizando, para esto se hace uso de la función cv2.CAP\_PROP\_FPS.

**4.** Método read.

def read(self):

return self.Q.get()

El método read manejara la devolución del siguiente frame en la cola del queue.

**5.** Método running.

def running(self):

return self.more() or not self.stopped

El método running se utiliza para cubrir el caso en que el programa haya alcanzado el final del video, de no ser así devuelve la instancia que corresponda.

**6.** Método more.

def more(self):

tries = 0

while self.Q.qsize() == 0 and not self.stopped and tries < 5:

time.sleep(0.1)

tries += 1

return self.Q.qsize() > 0

El método more devuelve un valor booleano de verdadero si todavía existen frames por leer de la estructura del queue. Si la ejecución del video no se ha detenido, da cinco intentos para agregar un tiempo de descanso a la ejecución del programa.

**7.** Método stop.

def stop(self):

self.stopped = True

self.thread.join()

El método stop será llamado si el usuario quiere detener el proceso de forma prematura (antes de llegar al final del video). En la última linea programa espera mientras los recursos de la ejecución del video son liberados, ya que el programa puede seguir grabando los frames.

**Código Centroidtracker.py**

El siguiente código se utiliza para hacer el seguimiento de los deportistas en el video a través del centroide de los recuadros que los identifican.

**Librerías**

**1.** Librerías de paquetes de funciones necesarias.

from scipy.spatial import distance as dist

from collections import OrderedDict

import numpy as np

*scipy.spatial*: El paquete de scipy.spatial [34] de la biblioteca de SciPy [35] para Python, puede computar triangulaciones, diagramas Voronoi y envolventes convexas de un set de puntos al aprovechar la biblioteca de Qhull [36]. Además, contiene implementaciones de KDTree [37] para consultas del punto vecino más cercano (nearest-neighbor point) y utilidades para calcular distancias en diversos tipos de métricas, los cuales se encuentran en el submodulo scipy.spatial.distance [38].

*collections*: El módulo de collections [39] implementa tipos de datos de contenedores especializados, que brindan una alternativa a los de uso general de Python, como son, dict, list, set, y tuple. OrderedDict [40] es una subclase de diccionario que devuelve el orden en que las entradas del diccionario fueron añadidas.

**Constructor de la clase CentroidTracker**

class CentroidTracker():

def \_\_init\_\_(self, maxDisappeared=50):

self.nextObjectID = 0

self.objects = OrderedDict()

self.originRects = OrderedDict()

self.disappeared = OrderedDict()

self.maxDisappeared = maxDisappeared

En las anteriores líneas de código se define el constructor para la clase CentroidTracker, la cual, toma como único parámetro de entrada el número máximo de frames consecutivos en los que un objeto puede estar perdido/desaparecido antes de ser removido del seguidor.

El constructor construye cuatro variables de clase:

* nextObjectID: Contador usado para asignar un ID o identificador único a cada objeto detectado. En el caso de que un objeto desaparezca del video por un número de frames superior al establecido en maxDisappeared, un nuevo identificador será asignado.
* objects: Diccionario creado con la subclase OrderedDict. Utiliza el ID del objeto como la clave o Key y las coordenadas (x,y) del centroide como el valor.
* originRects: Diccionario que utiliza el ID del objeto como la clave y las coordenadas de los recuadros de los objetos detectados como el valor.
* disappeared: Mantiene el numero (valor) de frames consecutivos que un ID (clave del diccionario) ha sido marcado como perdido.
* maxDisappeared: Número máximo de frames consecutivos en los que un objeto puede estar perdido/desaparecido antes de ser removido del seguidor.

**1**. Método register.

El método register es el encargado de añadir o registrar los nuevos objetos al seguidor.

def register(self, centroid, rect):

self.originRects[self.nextObjectID] = rect

self.objects[self.nextObjectID] = centroid

self.disappeared[self.nextObjectID] = 0

self.nextObjectID += 1

El método register tiene como únicos parámetros de entrada las coordenadas del centroide y de los recuadros. En las primeras dos líneas de esta función se añaden dichos parámetros a los diccionarios de originRects y Objects respectivamente, haciendo uso del nextObjectID.

El número de veces que un objeto ha sido marcado como desaparecido se establece en cero en la tercera línea del código. Finalmente, se incrementa el contador de nextObjectID, de tal forma que, si un nuevo objeto aparece en el frame, será asociado a un nuevo y único identificador.

**2.** Método deregister.

El método deregister es el encargado de eliminar los objetos que han sido marcados como desaparecidos.

def deregister(self, objectID):

del self.originRects[objectID]

del self.objects[objectID]

del self.disappeared[objectID]

Este método elimina el ID del objeto de los diccionarios que han sido creados anteriormente, para esto hace uso de la palabra clave de Python, del.

**3.** Método get\_id.

El método get\_id se utiliza para guardar las coordenadas de los recuadros en su respectivo diccionario y con su respectivo ID. El bloque de código utilizado para esto es el mismo que se utiliza dentro del siguiente método (update) para calcular el centroide y actualizar las listas de datos. La razón para crear un método aparte con este fragmento de código es que, cuando se realizan los cálculos del centroide dentro del método de update, las coordenadas de los recuadros se pierden y no hay forma de devolver este parámetro y de asociarlo al ID utilizado.

def get\_id(self, rect):

(x, y, eX, eY) = rect

cX = ((x + eX) / 2.0)

cY = ((y + eY) / 2.0)

objectIDs = list(self.objects.keys())

objectCentroids = list(self.objects.values())

D = dist.cdist(np.array(objectCentroids), [(cX, cY)])

rows = D.min(axis=1).argsort()

cols = D.argmin(axis=1)[rows]

objectID = None

for (row, col) in zip(rows, cols):

objectID = objectIDs[row]

break

return objectID

La explicación del anterior bloque de código es la misma que se encuentra para el método update.

**4.** Método update.

El método update es el encargado principal de todo el proceso de seguimiento que hace el centroidtracker. El único parámetro de entrada de este método son las coordenadas de los recuadros de los objetos detectados, las cuales se esperan tengan el siguiente formato: (starX,starY,endX,endY).

def update(self, rects):

if(len(rects) == 0):

for objectID in list(self.disappeared.keys()):

self.disappeared[objectID] += 1

if(self.disappeared[objectID] > self.maxDisappeared):

self.deregister(objectID)

return self.objects, self.originRects

Si no se han realizado detecciones, es decir si rects = 0, se itera sobre todos los ID existentes y se incrementa su contador de disappeared. De igual forma, se comprueba si se ha alcanzado el número máximo de maxDisappeared y, de ser el caso, se utiliza el método deregister para eliminar dicho ID del seguidor. Ya que no hay información del seguidor para actualizar, se retornan los diccionarios de objects y originRects.

En el siguiente bloque de código se inicializa un array de numpy para almacenar los centroides calculados para cada recuadro en rects. Para crear el array se utiliza la función np.zeros [41], la cual retorna un array del tamaño ((len(rects), 2) y tipo (‘int’) dados, relleno con ceros.

inputCentroids = np.zeros((len(rects), 2), dtype="int")

for(i, (startX, startY, endX, endY)) in enumerate(rects):

cX = int((startX + endX) / 2.0)

cY = int((startY + endY) / 2.0)

inputCentroids[i] = (cX, cY)

Luego de esto, se itera con un ciclo for a través de las coordenadas de los recuadros, las cuales se enumeran mediante la función enumerate () de Python [42]. Con las coordenadas de los recuadros, se procede a calcular las coordenadas de los centroides y a almacenarlas en el array previamente inicializado.

Si no se está haciendo seguimiento a algún objeto, el programa pasa a registrar las coordenadas del centroide y del recuadro de cada uno de los nuevos objetos.

if(len(self.objects) == 0):

for i in range(0, len(inputCentroids)):

centroid = inputCentroids[i]

rect = rects[i]

self.register(centroid, rect)

En la cuarta línea del anterior fragmento de código se itera sobre las coordenadas de los recuadros que se pasan al método de update (rects) y se asignan a una nueva variable, rect, la cual se utiliza en el método de register y get\_id para hacer el respectivo registro de estos datos.

Por otra parte, si se necesitan actualizar las coordenadas (x,y) de un objeto existente, basado en la locación del centroide que minimice la distancia euclidiana entre ellos, se realiza lo siguiente:

else:

objectIDs = list(self.objects.keys())

objectCentroids = list(self.objects.values())

D = dist.cdist(np.array(objectCentroids), inputCentroids)

rows = D.min(axis=1).argsort()

cols = D.argmin(axis=1)[rows]

En las primeras dos líneas de código que siguen al else, se guardan las claves y valores del diccionario de objects, que corresponden respectivamente, a los ID de los objetos y a sus centroides.

El objetivo de este fragmento de código es realizar el seguimiento a los objetos manteniendo los ID asignados en un principio a cada uno, esto se logra al calcular la distancia euclidiana entre todos los pares de centroides ya registrados (objectCentroids) y todos los nuevos centroides (inputCentroids), para hacer este calculo se hace uso de la función dist.cdist [43] de la librería scipy.spatial. El tamaño del array Numpy de salida del mapa de distancia D será (# de centroides de objetos, # de centroides de entrada).

Para realizar el emparejamiento entre ambos pares de centroides se debe:

1. Encontrar el valor de distancia más pequeño en cada fila y clasificar los índices de estas basado en el minimo valor encontrado. Esto se logra con el método min y la función argsort() [44], la cual devuelve los índices que ordenarían el array de acuerdo a la clasificación realizada.
2. Se realiza un proceso similar en las columnas del array D, encontrando el valor más pequeño y clasificándolos luego basado en el orden de las filas.

El objetivo de esto es tener los índices de los valores con la distancia más pequeña al inicio de las listas.

El siguiente paso es usar las distancias para ver si se pueden asociar los IDs.

usedRows = set()

usedCols = set()

for (row, col) in zip(rows, cols):

if row in usedRows or col in usedCols:

continue

objectID = objectIDs[row]

self.objects[objectID] = inputCentroids[col]

self.originRects[objectID] = rects[col]

self.disappeared[objectID] = 0

usedRows.add(row)

usedCols.add(col)

Dentro del anterior bloque de código se realiza lo siguiente:

1. Se inicializan dos sets para determinar cuáles índices de filas y columnas han sido utilizados. Se debe tomar en cuenta que un set [45] es similar a una lista pero con la diferencia de que el primero contiene solo valores únicos.
2. Se itera sobre las combinaciones de índices de las tuplas (rows,cols) para actualizar los centroides de los objetos.

* Si ya se ha utilizado el índice ya sea para fila o para columna, se ignora y se continua con el ciclo.
* De otra forma, si se ha encontrado un centroide que tiene la mínima distancia euclidiana a un centroide ya existente y no ha sido asociado a algún otro objeto, se actualiza el centroide en los diccionarios correspondientes y se añaden los índices de la fila y la columna a sus sets respectivos.

Ya que es probable que existan índices en los sets de usedRows y usedCols que no hayan sido examinados todavía, se utiliza el siguiente bloque de código para determinar cuáles índices no han sido examinados y almacenarlos en dos nuevos sets.

unusedRows = set(range(0, D.shape[0])).difference(usedRows)

unusedCols = set(range(0, D.shape[1])).difference(usedCols)

Para almacenar solo aquellos valores que no se hayan examinado, se utiliza el método difference [46] de Python, el cual devuelve la diferencia entre los dos sets dados.

Los últimos dos bloques de código se utilizan para manejar cualquier objeto que haya sido marcado como desaparecido.

if D.shape[0] >= D.shape[1]:

for row in unusedRows:

objectID = objectIDs[row]

self.disappeared[objectID] += 1

if self.disappeared[objectID] > self.maxDisappeared:

self.deregister(objectID)

Para finalizar:

* Si el numero de centroides actuales es mayor o igual que el número de centroides de entrada:
* Es necesario verificar si alguno de estos objetos se ha desaparecido, esto se hace al iterar sobre los índices de filas no utilizados.
* En el ciclo for:

1. Se incrementará el contador de disappeared en el diccionario
2. Se verificará si el contador de disappeared excede el valor máximo de maxdissapeared y, si es así, se eliminará dicho objeto del diccionario.

* De otra forma, si el número de centroides de entrada es mayor que el número de centroides existentes significa que hay un nuevo objeto por registrar.

else:

for col in unusedCols:

centroid = inputCentroids[col]

rect = rects[col]

self.register(centroid, rect)

return self.objects, self.originRects

Para esto, se itera sobre los índices de columnas no utilizados y se registra cada nuevo centroide y recuadro. Finalmente, se retornan los sets de objetos rastreables al método de llamada.

**Código Selección\_ROI.py**

El siguiente código se utiliza para hacer el mapeo del escenario deportivo y a su vez almacenar los puntos que contiene las coordenadas del mismo.

**Librerías**

**1.** Librerías de paquetes de funciones necesarias.

import cv2

import os

import numpy as np

import pickle

*pickle*: El módulo pickle [47] implementa protocolos binarios para serializar y deserializar una estructura de objetos Python. «Pickling» es el proceso mediante el cual una jerarquía de objetos de Python se convierte en una secuencia

**2.** Método def get\_mouse\_points

Inicialmente se realiza selección de los cuatro puntos de origen de manera interactiva a través de la captura de los eventos del mouse en la función def get\_mouse\_points()

def get\_mouse\_points(event, x, y, flags, param):

global mouseX, mouseY, mouse\_pts

if event == cv2.EVENT\_LBUTTONDOWN:

mouseX, mouseY = x, y

if len(mouse\_pts)<=3:

cv2.circle(frame, (x, y), 7, (0, 255, 255), 6,cv2.LINE\_AA)

else:

cv2.circle(frame, (x, y), 7, (0, 0, 255), 6,cv2.LINE\_AA)

mouse\_pts.append((x, y))

Para la extracción de los puntos que dibujaran el escenario deportivo, se determina si el mouse presenta un evento (click) mediante el comando cv2.EVENT\_LBUTTONDOWN, dado el caso, que se cumpla la condición, se procede a registrar las coordenadas (X, Y) y a su vez dibújalas mediante un círculo. Para realizar el dibujo se utilizó el comando cv2.circle(), cabe resaltar que dicha condición, solo realzara el registro de 4 puntos, que son los dibujan el recuadro en el que se realiza el test. Estos puntos serán almacenados en una lista, la cual será utilizada para generar un archivo con La lista de puntos.

**Ciclo While**

Luego de haber importado e inicializado las variables y funciones necesarias para el procesamiento del programa, se crea un ciclo while, el cual va a iterar sobre el buffer de frames que se han leído del video, para esto, se crea un valor booleano que permanece verdadero mientras existan frames almacenados en el buffer sobre los cuales se pueda iterar, al hacer uso de esta condición en el ciclo while se asegura de que el ciclo se va a seguir repitiendo mientras hayan frames por procesar, en el momento en que el buffer se quede sin frames o se oprima la tecla encargada de cerrar el ciclo, el valor booleano cambia a falso y el ciclo while se rompe.

**1**. Lectura de frames.

Lo primero que se realiza dentro del ciclo while es la lectura de un contador que nos va a permitir obtener el frame 0, el cual será utilizado para dibujar los 4 puntos del escenario deportivo. Seguido a esto, se utiliza un ciclo if para el determinado caso de que la variable frame se encuentre vacía, de ser así el ciclo se rompe.

while True:

frame\_num += 1

ret, frame = cap.read()

if not ret:

break

**2.** Ciclo while

Si el contador de frames es 1, entrará al ciclo while, el cual será el encargado de la determinación de los puntos que contemplaran el escenario deportivo, este ciclo se cerrara, siempre y cuando se seleccionen los 4 puntos o se presione la tecla de “Esc”.

if frame\_num == 1:

while True:

cv2.imshow("Seleccion ROI", frame)

key = cv2.waitKey(1) & 0xFF

if key == 27:

break

if len(mouse\_pts) == 4:

cv2.destroyWindow("Seleccion ROI")

#select = True

break

four\_points = mouse\_pts[0:4]

cv2.imwrite('Definicion\_ROI.png',frame)

**3.** condición if

Siempre y cuando se hallan seleccionado los 4 puntos, se procede a realizar el rectángulo del escenario, mediante la función cv2.polylines() la cual tendrá como parámetros de entrada, el video, los puntos, el color, el tipo de línea y el llenado. Por otro lado, si la selección fue nula se procede a informar de que no fue realizada la selección.

if len(four\_points ) == 4:

pts = np.array([four\_points[0], four\_points[1], four\_points[3], four\_points[2]], np.int32)

thickness=3

cv2.polylines(frame, [pts], True, (0, 255, 255), thickness, cv2.LINE\_AA)

else:

print('[INFO]No fue realizada seleccion!')

**4.** guardado de archivo

Finalizado el ciclo se procede a guardar un archivo con los puntos que contienen la región de interés para el escenario deportivo.

with open('puntos.data', 'wb') as filehandle:

pickle.dump(four\_points, filehandle)

print('[INFO]Seleccion guardada!')

cap.release()

cv2.destroyAllWindows()

Con la función cv2.destroyAllWindows [25] se destruyen todas las ventanas abiertas durante el procesamiento del programa.

**Código aux\_functions.py**

El siguiente código es el encargado de obtener la vista de arriba hacia abajo o vista de ojos de pájaro loa cual nos permite tener el escenario deportivo para la realización del test. Para la transformación de perspectiva se necesita una matriz de transformación 3x3.

**Librerías**

Librerías de paquetes de funciones necesarias.

import cv2

import os

import numpy as np

import pickle

La siguiente función tendrá como parámetros de entrada el frame y los puntos capturados anteriormente, para la representación del escenario deportivo. Posteriormente se obtienen las dimensiones del video, para la creación de un arreglo de tipo float32, la cual tendrá el mismo orden que los puntos definidos por la selección ROI del escenario a evaluar, seguido a esto se procede a crear un arreglo numpy con la lista de los 4 puntos y esta tiene que ser de tipo float32.

def get\_camera\_perspective(img, src\_points):

IMAGE\_H = img.shape[0]

IMAGE\_W = img.shape[1]

src = np.float32(np.array(src\_points))

dst = np.float32([[0, IMAGE\_H], [IMAGE\_W, IMAGE\_H], [0, 0], [IMAGE\_W, 0]])

M = cv2.getPerspectiveTransform(src, dst)

M\_inv = cv2.getPerspectiveTransform(dst, src)

return M, M\_inv

Para obtener realmente la vista de arriba hacia abajo, "vista de ojos de pájaro" de la imagen vamos a utilizar la función cv2.getPerspectiveTransform() que requiere dos argumentos, la lista de 4 puntos del ROI en la imagen original y la lista de puntos de destino, definidos a partir del orden específico para los puntos de origen, danos como resultado una matriz de transformación real M, la cual será retornada para su respectivo cambio de perspectiva en el código Principal.

**Referencias**

1. https://numpy.org/
2. https://opencv.org/
3. https://docs.python.org/3/library/math.html
4. https://docs.python.org/3/library/time.html
5. https://github.com/facebookresearch/detectron2/blob/master/MODEL\_ZOO.md
6. https://detectron2.readthedocs.io/modules/utils.html?highlight=detectron2.utils.logger#detectron2.utils.logger.setup\_logger
7. https://detectron2.readthedocs.io/modules/engine.html?highlight=detectron2.engine.%20DefaultPredictor#detectron2.engine.defaults.DefaultPredictor
8. https://detectron2.readthedocs.io/modules/config.html?highlight=detectron2.config.get\_cfg#detectron2.config.get\_cfg
9. https://detectron2.readthedocs.io/modules/utils.html?highlight=detectron2.utils.visualizer.Visualizer#detectron2.utils.visualizer.Visualizer
10. https://detectron2.readthedocs.io/modules/data.html?highlight=detectron2.data.MetadataCatalog#detectron2.data.MetadataCatalog
11. https://detectron2.readthedocs.io/modules/structures.html?highlight=detectron2.structures.Boxes#detectron2.structures.Boxes
12. https://detectron2.readthedocs.io/modules/structures.html?highlight=detectron2.structures.BoxMode#detectron2.structures.BoxMode
13. https://detectron2.readthedocs.io/modules/structures.html?highlight=detectron2.structures.pairwise\_iou#detectron2.structures.pairwise\_iou
14. https://detectron2.readthedocs.io/modules/model\_zoo.html?highlight=cfg.merge\_from\_file.model\_zoo.get\_config\_file#detectron2.model\_zoo.get\_config\_file
15. https://detectron2.readthedocs.io/modules/model\_zoo.html?highlight=model\_zoo.get\_checkpoint\_url#detectron2.model\_zoo.get\_checkpoint\_url
16. https://detectron2.readthedocs.io/modules/engine.html?highlight=DefaultPredictor(cfg)#detectron2.engine.defaults.DefaultPredictor
17. https://docs.opencv.org/3.4/dd/d9e/classcv\_1\_1VideoWriter.html#ad59c61d8881ba2b2da22cff5487465b5
18. http://www.fourcc.org/codecs.php
19. https://docs.python.org/3.3/library/stdtypes.html?highlight=range#ranges
20. https://docs.opencv.org/master/d6/d6e/group\_\_imgproc\_\_draw.html#ga07d2f74cadcf8e305e810ce8eed13bc9
21. https://docs.opencv.org/master/d6/d6e/group\_\_imgproc\_\_draw.html#ga5126f47f883d730f633d74f07456c576
22. https://docs.opencv.org/master/d6/d6e/group\_\_imgproc\_\_draw.html#gaf10604b069374903dbd0f0488cb43670
23. https://docs.opencv.org/master/d4/da8/group\_\_imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce
24. https://docs.opencv.org/master/d7/dfc/group\_\_highgui.html#ga5628525ad33f52eab17feebcfba38bd7
25. https://docs.opencv.org/master/d7/dfc/group\_\_highgui.html#ga6b7fc1c1a8960438156912027b38f481
26. Queue pdf.
27. https://docs.python.org/3/library/threading.htm
28. https://docs.python.org/3/library/threading.html#threading.Thread
29. https://docs.python.org/3/library/sys.html
30. https://docs.python.org/3/library/queue.html?highlight=queue#queue.Queue
31. https://docs.python.org/3/library/queue.html?highlight=queue#module-queue
32. https://docs.python.org/2/library/sys.html#sys.version\_info
33. https://docs.opencv.org/3.4/d8/dfe/classcv\_1\_1VideoCapture.html#a57c0e81e83e60f36c83027dc2a188e80
34. https://scipy.github.io/devdocs/tutorial/spatial.html
35. https://www.scipy.org/
36. http://www.qhull.org/
37. https://scipy.github.io/devdocs/generated/scipy.spatial.KDTree.html#scipy.spatial.KDTree
38. https://docs.scipy.org/doc/scipy/reference/spatial.distance.html#module-scipy.spatial.distance
39. https://docs.python.org/3/library/collections.html
40. https://docs.python.org/3/library/collections.html#collections.OrderedDict
41. https://numpy.org/doc/stable/reference/generated/numpy.zeros.html
42. https://docs.python.org/3/library/functions.html#enumerate
43. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html
44. https://numpy.org/doc/stable/reference/generated/numpy.argsort.html
45. https://docs.python.org/2/library/sets.html
46. https://beginnersbook.com/2019/03/python-set-difference-method/#:~:text=The%20difference()%20method%20in,A%20but%20in%20Set%20B.
47. https://docs.python.org/es/3/library/pickle.html#module-pickle.