

TRABAJO FINAL VC

VIRTUALIZACIÓN DE UN TABLERO DE AJEDREZ

CARLOS TOMÁS QUEVEDO OLIVARES



ULPGC
Universidad de
Las Palmas de
Gran Canaria

Índice

INTRODUCCIÓN	3
OBJETIVO	4
TECNOLOGÍAS UTILIZADAS.....	5
DESARROLLO	6
Homografía	6
Detección y Posición	10
Virtualización	16
CONCLUSIÓN Y PROPUESTAS DE AMPLIACIÓN.....	21

INTRODUCCIÓN

Actualmente y desde hace mucho tiempo la tecnología empleada para reproducir partidas de ajedrez a nivel competitivo consiste en la utilización de tableros electrónicos de alto coste con sensores físicos para registrar el movimiento y la posición de las piezas. Es por ello por lo que, resulta interesante realizar una aproximación alternativa mediante el uso de visión por computador para este propósito.

OBJETIVO

Para el desarrollo de esta práctica me he centrado fundamentalmente en la detección y virtualización de una posición correspondiente a una imagen estática en un tablero de ajedrez, para ello decidí enfocarme necesariamente en cuatro aspectos principales.

- Homografía del tablero: establecimiento de una relación entre el tablero que se quiere virtualizar y su equivalencia respecto a un tablero normalizado desde una perspectiva cenital.
- Detección de las piezas: detección y clasificación de las piezas que se encuentran en el tablero.
- Posición de la piezas en el tablero: una vez realizados los dos puntos anteriores conocer la casilla en la que se encuentra cada pieza.
- Virtualización: en último lugar virtualización de la posición

TECNOLOGÍAS UTILIZADAS

Para el desarrollo de esta práctica he utilizado YOLO para el entrenamiento de un modelo capaz de detectar y clasificar las diferentes piezas de ajedrez además de, las siguientes librerías de Python:

- cv2 (OpenCV): Proporciona herramientas para procesamiento de imágenes y visión por computadora, como carga, manipulación y análisis de imágenes.
- YOLO (Ultralytics): Necesaria para manipular un modelo entrenado para la detección de piezas de ajedrez basado en la red neuronal YOLO.
- matplotlib.pyplot: Permite crear gráficos y visualizar datos, como imágenes, histogramas o gráficos de dispersión.
- numpy (np): Proporciona estructuras de datos y operaciones avanzadas para cálculos matemáticos y manipulación de matrices.
- skimage.transform: Ofrece herramientas para transformar imágenes, como redimensionar, rotar, escalar y ajustar perspectivas(homografía).
- pygame: Librería para el desarrollo de videojuegos en Python, proporcionando herramientas para gráficos 2D, sonido, manejo de eventos y control de usuarios.
- chess: Biblioteca para crear y manipular juegos de ajedrez, permitiendo implementar movimientos legales, análisis de posiciones, generación de tableros y soporte para formatos estándar como PGN y FEN.

```
import cv2
from ultralytics import YOLO
from matplotlib import pyplot as plt
import numpy as np
from skimage import transform
```

```
import pygame
import chess
```

DESARROLLO

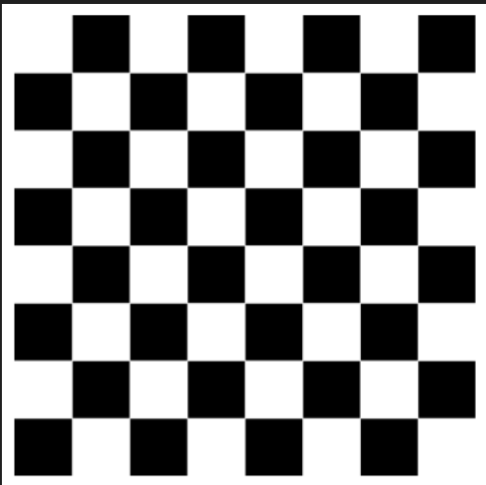
Homografía

En primer lugar generamos un tablero de ajedrez normalizado haciendo uso de las herramientas de openCV.

Decidí generarlo de esta forma en lugar de sacarlo de internet puesto que, así conozco previamente las dimensiones del tablero y por extensión las coordenadas de cada casilla, esto resultara muy útil en el futuro tanto para realizar la homografía, como para conocer la posición de las piezas.

```
#Generamos un tablero de ajedrez el cual vamos a tomar de referencia para realizar la homografía
tablero = np.zeros((800,800,3), dtype = np.uint8)
for filas in range(8):
    for columnas in range(8):
        if ((filas + columnas) % 2) == 0:
            cv2.rectangle(tablero, (filas*100, columnas*100), ((filas+1)*100, (columnas+1)*100), (255,255,255), -1)
plt.axis("off")
plt.imshow(tablero)
plt.show()
```

✓ 0.0s



En segundo lugar, obtenemos las coordenadas correspondientes a las esquinas de cada tablero, las del tablero que acabamos de crear ya las conocemos previamente por lo que simplemente las guardamos, por otra parte las coordenadas del tablero que queremos detectar las

obtenemos de forma manual clicando en las correspondientes esquinas del tablero. El orden para hacerlo(muy importante) seria(desde una perspectiva de las blancas), esquina inferior izquierda, inferior derecha, superior izquierda, superior derecha.

```
##### 1. RECOPILA LOS CUATRO PUNTOS EN AMBAS IMÁGENES
#Inicializa listas depuntos
puntosA = []
puntosB = [(0, 800), (800, 800), (0, 0), (800, 0)]

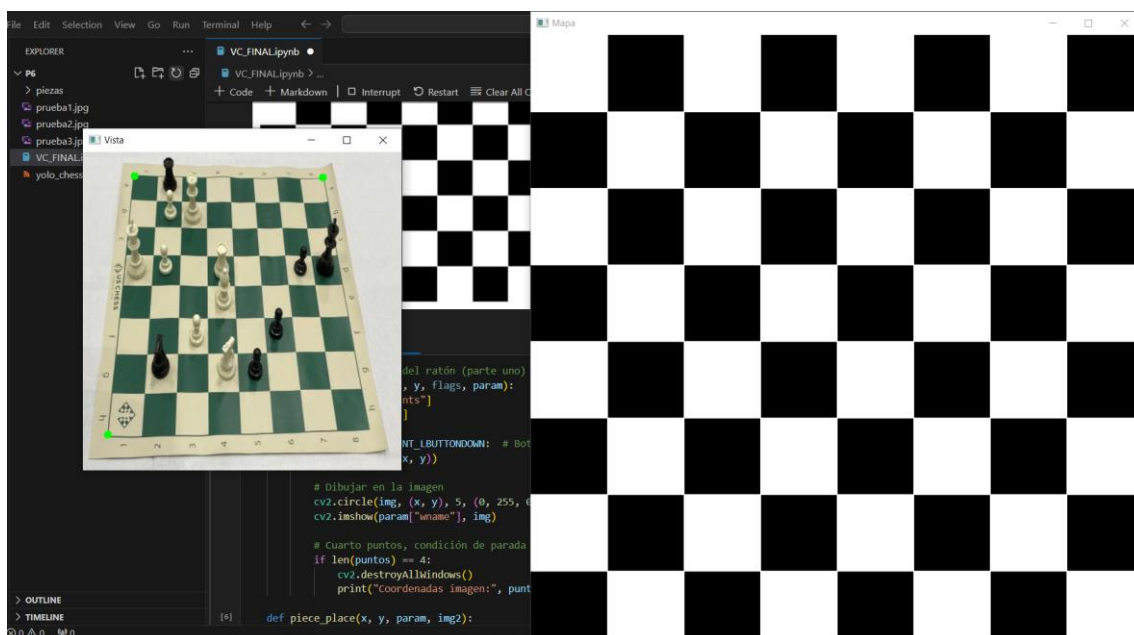
#Lee la nueva imagen, crea copia de trabajo
vista = cv2.imread('prueba3.jpg')
vistatmp = vista.copy()

#Mostramos el tablero de ajedrez de referencia
cv2.imshow('Mapa', tablero)
cv2.waitKey(0)

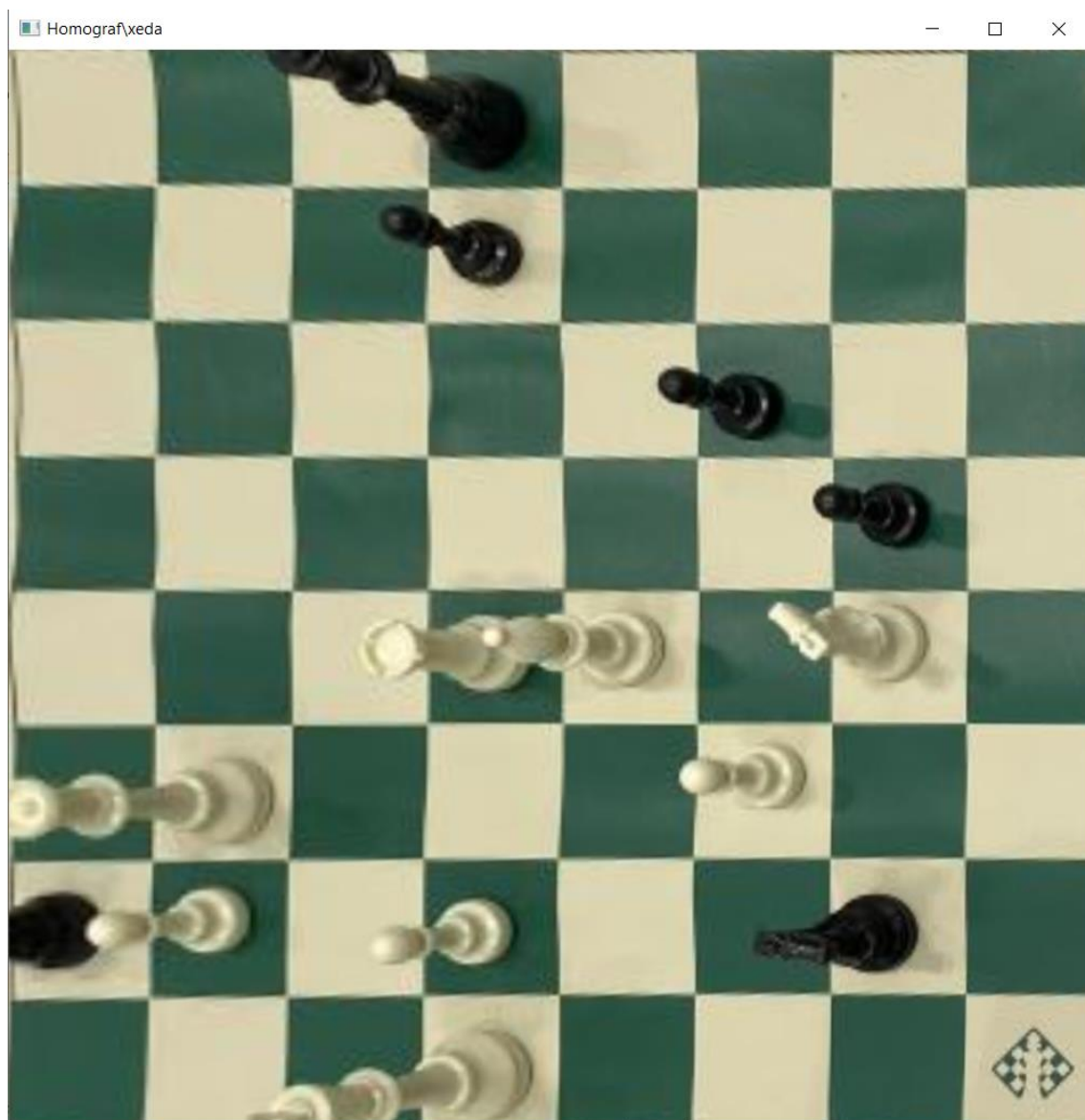
#Vista de la posicion a virtualizar
cv2.imshow("Vista", vistatmp)
params = {
    "points": puntosA,
    "image": vistatmp,
    "wname": "Vista"
}
cv2.setMouseCallback("Vista", get_points, params)
# Selecciona cuatro puntos o cierra ventana
cv2.waitKey(0)
cv2.destroyAllWindows()

#Transformación de los puntos https://scikit-image.org/docs/stable/auto\_examples/transform/plot\_transform\_types.html
tform = transform.estimate_transform('projective', np.array(puntosA), np.array(puntosB))
tf_img = transform.warp(vista, tform.inverse, output_shape=(tablero.shape))

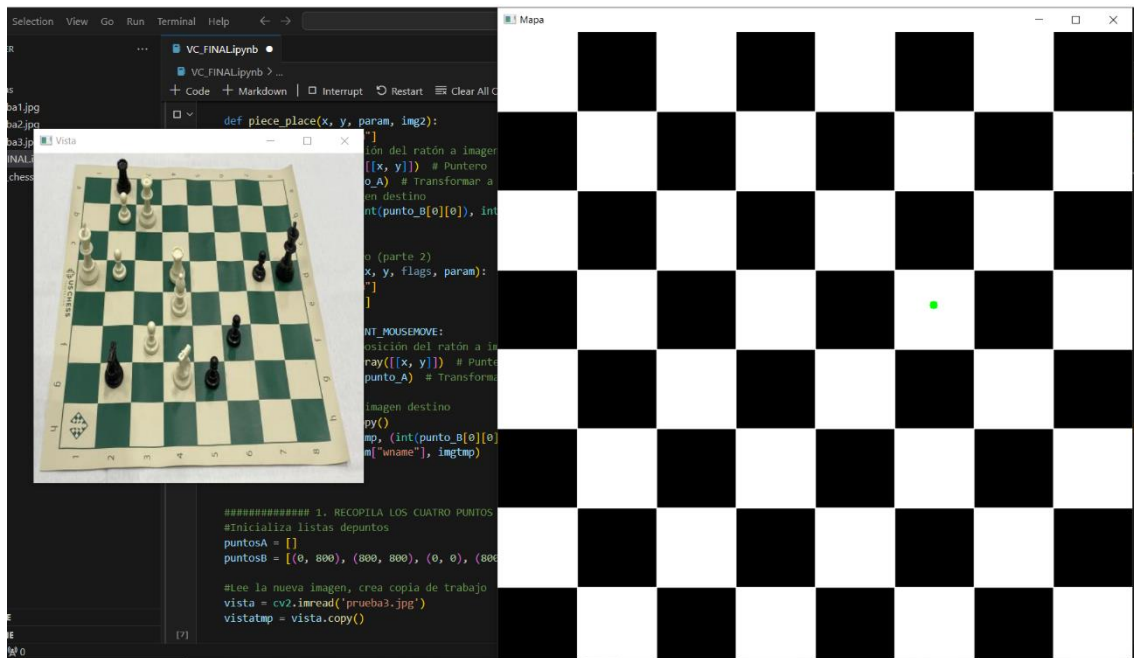
#Muestra imagen de entrada transformada
cv2.imshow("Homografía", tf_img)
cv2.waitKey(-1)
cv2.destroyAllWindows()
```



Así quedaría la imagen generada de la vista del tablero desde un plano cenital.



Con la transformación realizada, podemos movernos por el tablero objeto de estudio y comprobar su posición correspondiente en el tablero normalizado.



Detección y Posición

Para realizar la detección de las piezas de ajedrez en el tablero debemos entrenar un modelo para ello. Al igual que en ocasiones anteriores y aprovechando la misma configuración, he hecho uso de YOLO, utilizando la GPU para el entrenamiento.

El dataset correspondiente al entrenamiento del modelo fue extraído de la página roboflow y se puede obtener en el siguiente enlace.

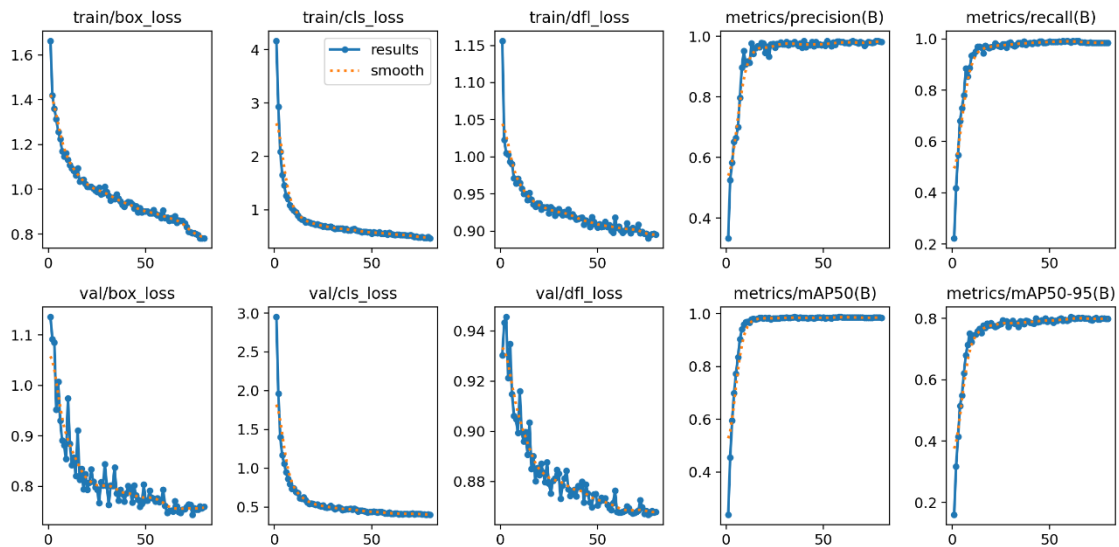
<https://universe.roboflow.com/joseph-nelson/chess-pieces-new/dataset/24>

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:18<00:00,
1/80	0.34G	1.663	4.168	1.156	19	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 14.16it/s]
	Class	Images	Instances	Box(P	R	0.239	0.16		
	all	58	386	0.333	0.222				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:14<00:00, 10.72it/s]
2/80	0.338G	1.42	2.929	1.023	49	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 17.90it/s]
	Class	Images	Instances	Box(P	R	0.454	0.318		
	all	58	386	0.526	0.417				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:13<00:00, 11.11it/s]
3/80	0.333G	1.361	2.093	1.005	34	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 18.18it/s]
	Class	Images	Instances	Box(P	R	0.595	0.414		
	all	58	386	0.583	0.548				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:13<00:00, 10.92it/s]
4/80	0.325G	1.313	1.657	1.003	22	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 16.49it/s]
	Class	Images	Instances	Box(P	R	0.699	0.515		
	all	58	386	0.652	0.68				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:14<00:00, 10.72it/s]
5/80	0.327G	1.257	1.46	0.9934	21	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 16.49it/s]
	Class	Images	Instances	Box(P	R	0.773	0.549		
	all	58	386	0.664	0.73				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:13<00:00, 11.11it/s]
6/80	0.361G	1.226	1.268	0.9906	33	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 16.16it/s]
	Class	Images	Instances	Box(P	R	0.837	0.621		
	all	58	386	0.701	0.78				
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	416: 100%	152/152	[00:13<00:00, 11.09it/s]
7/80	0.333G	1.171	1.205	0.971	19	mAP50	mAP50-95): 100%	8/8	[00:00<00:00, 17.54it/s]
	Class	Images	Instances	Box(P	R	0.903	0.68		
	all	58	386	0.798	0.885				

Desde las primeras épocas los resultado apuntan muy bien, la detección de las cajas parece ser muy precisa según la variable mAP50, por otra parte, box_loss, cls_lss y dfl_lss que hacen referencia a la posición de las cajas que se detectan y a la clasificación del objeto en su interior, bajan de forma continua según avanza el entramiento.

El entrenamiento fue de 80 épocas pues hasta entonces los resultados fueron mejorando. A la hora de probar el modelo este me proporcione detecciones bastante buenas por lo que decidí escogerlo para la práctica.

Las siguientes gráficas muestran la mejora continua de los indicadores del entrenamiento comentados anteriormente para las 80 épocas.



Una vez listo el modelo, procedemos a realizar la detección sobre la imagen que queremos estudiar, las imágenes seleccionadas para hacer las pruebas fueron extraídas de la carpeta test del dataset. Evidentemente, la imagen sobre la que realizamos la homografía y sobre la que realizamos la detección debe coincidir.

Inicializamos el modelo y creamos una matriz vacía de 8x8 donde guardaremos la pieza detectada y su posición en el tablero indicada por el índice de la matriz donde guardaremos la pieza.

```
model = YOLO('yolo_chess_pieces.pt') #Contenedores

# Etiqueta de las distintas clases
classNames = ['bishop', 'black-bishop', 'black-king', 'black-knight', 'black-pawn', 'black-queen', 'black-rook', 'white-bishop', 'white-king', 'white-knight', 'white-pawn', 'white-queen', 'white-rook']

## Leer la imagen
img2 = cv2.imread('prueba3.jpg') # Cambia esto por la ruta de tu imagen

# Tablero de Referencia
tablero_cenital = tablero.copy()

# Procesar con el modelo de piezas de ajedrez
results = model(img2)

# Crear una matriz 8x8 con listas de listas
posicion = [['' for _ in range(8)] for _ in range(8)]
```

Realizamos lo descrito anteriormente, es decir, guardamos la pieza en sus respectiva posición en la matriz, pasando las coordenadas de la caja detectada en la imagen original por una función que la transforma en su posición correspondiente en el tablero normalizado. Esto es posible gracias a lo realizado en el primer punto de la práctica. Dependiendo de las coordenadas obtenidas gracias a esa función se

tratará de una casilla u otra y podremos guardarla en la matriz vacía en su lugar correspondiente.

```
# Procesar las detecciones
for r in results:
    boxes = r.bboxes
    for box in boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0]) # Coordenadas de la caja delimitadora
        cls = int(box.cls[0]) # Clase detectada (pieza de ajedrez)

        escala = int((cls / len(classNames)) * 255 * 3)
        R, G, B = (255, 255, escala - 255*2) if escala >= 255*2 else (255, escala - 255, 0) if escala >= 255 else (escala, 0, 0)

        # Dibuja y etiqueta la pieza detectada
        cv2.rectangle(img2, (x1, y1), (x2, y2), (R, G, B), 3)
        cv2.putText(img2, f"{classNames[cls]}", (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, B), 2)

        # Guarda la posición de la pieza y su clase en una matriz
        punto_B = piece_place(x1+5, y2-5, params, tablero_cenital)
        y = int((punto_B[0][0] // 100)) % 10 # Divide entre 100, luego toma el residuo de 10
        x = int((punto_B[0][1] // 100) % 10) # Divide entre 100, luego toma el residuo de 10
        posicion[x][y] = classNames[cls]

print(posicion)
```

Finalmente como paso previo al último apartado, aprovechamos y transformamos la matriz de posiciones obtenida en una línea de texto de formato fen. El formato fen es un tipo de anotación ajedrecística legible para librerías de programación como chess en Python, con la que se generara la posición resultado de la detección.

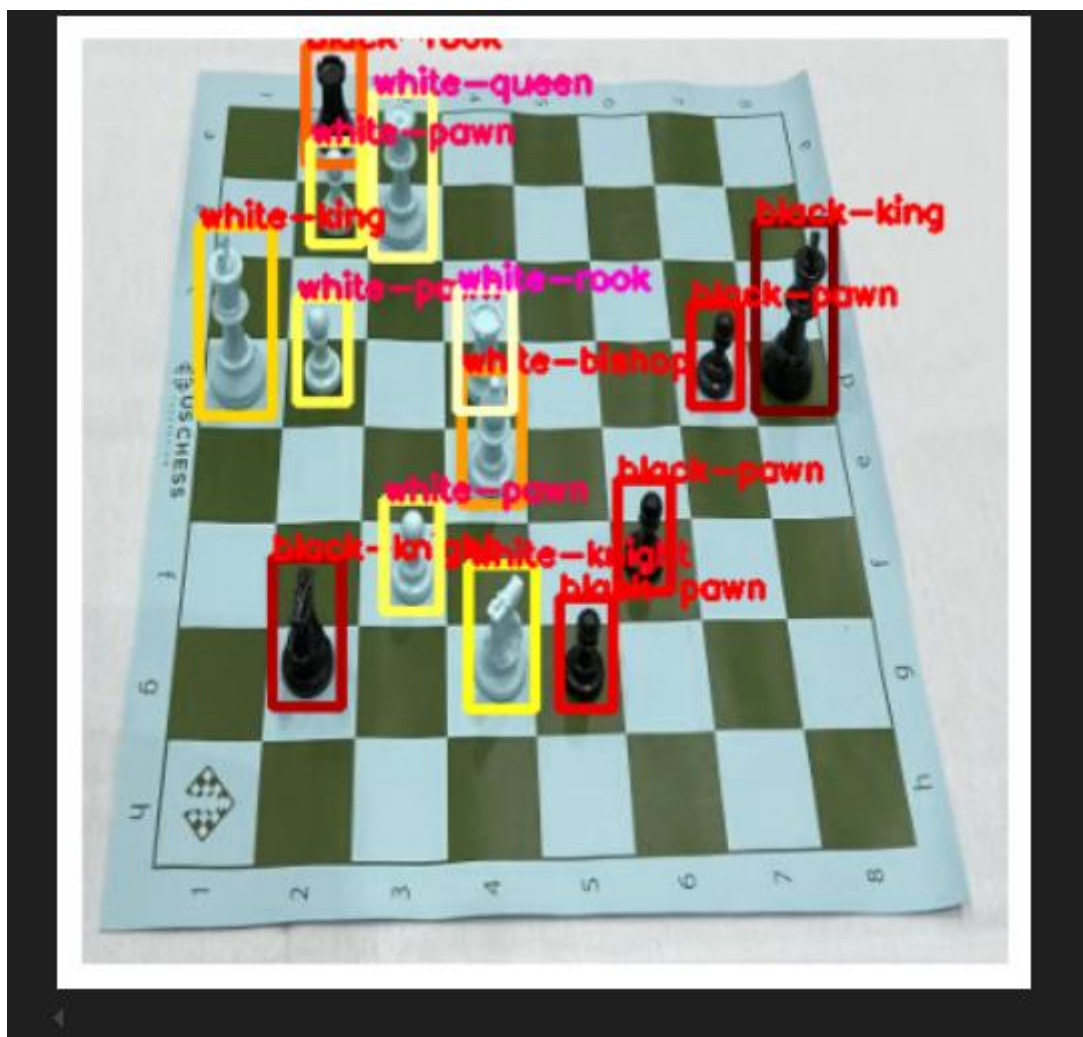
```
# Funcion para tranformar la matriz de posiciones y piezas a formato FEN
def board_to_fen(board_matrix):
    fen = []
    for row in board_matrix:
        row_fen = ''
        empty_count = 0
        for square in row:
            if square == ' ': # Casilla vacía
                empty_count += 1
            else: # Casilla ocupada por una pieza
                if empty_count > 0:
                    row_fen += str(empty_count) # Añadir número de casillas vacías
                    empty_count = 0
                # Convertir las piezas al formato estándar FEN
                if (square.find('-') + 1) == 'k' and square[square.find('-') + 2] == 'n':
                    piece = square.split('-')[1][1] # Tomar la primera letra después del guion
                else:
                    piece = square.split('-')[1][0] # Tomar la primera letra después del guion
                if 'white' in square:
                    piece = piece.upper() # Piezas blancas en mayúsculas
                else:
                    piece = piece.lower() # Piezas negras en minúsculas
                row_fen += piece
            if empty_count > 0:
                row_fen += str(empty_count) # Añadir número de casillas vacías al final de la fila
        fen.append(row_fen)
    return '/'.join(fen)

# Convertir la matriz al formato FEN
fen = board_to_fen(posicion)
print(fen)
```

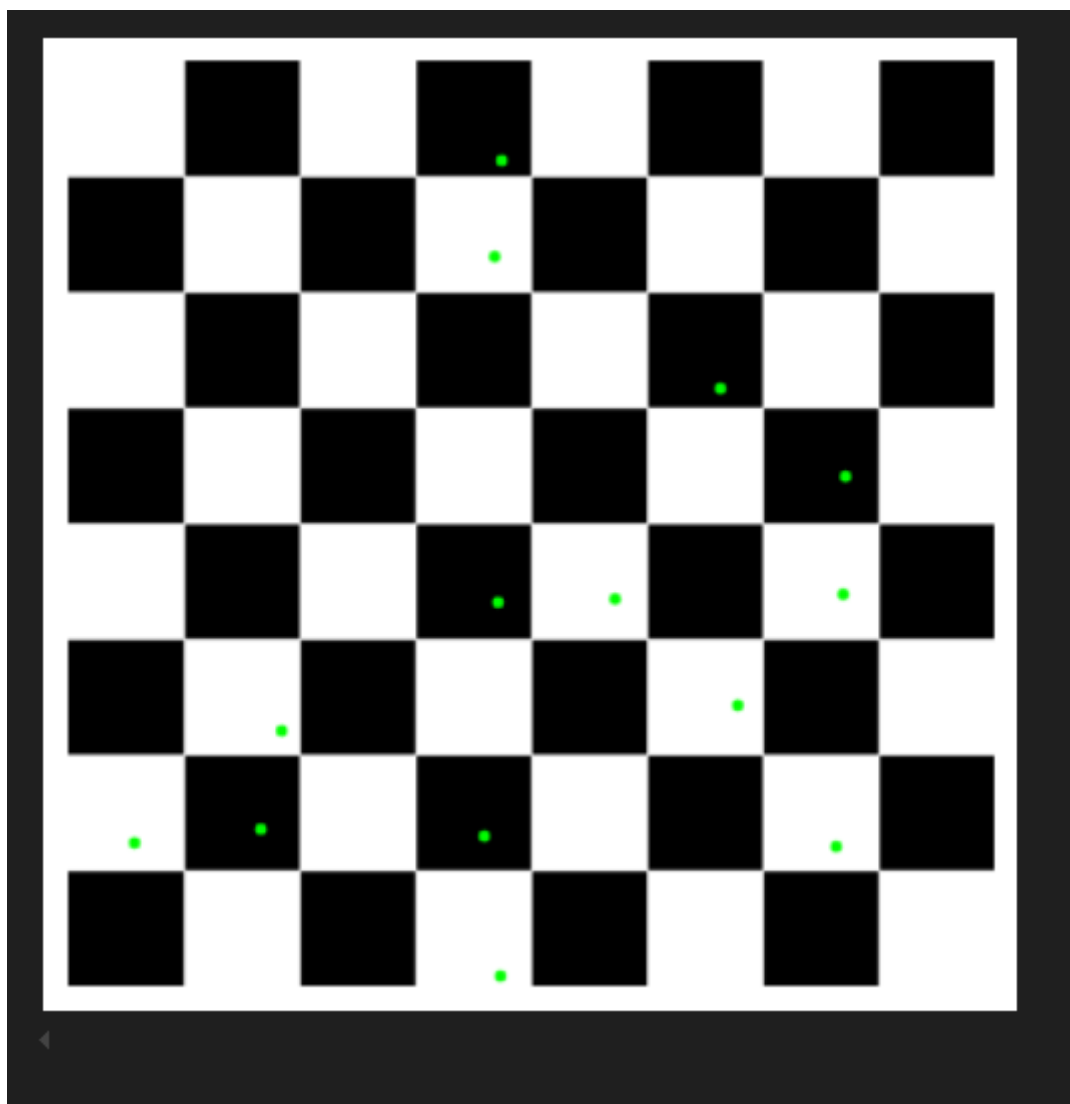
Esta sería la imagen que refleja la salida de las piezas detectadas, la matriz de posición y la línea de texto en formato fen.

```
0: 416x416 1 black-king, 1 black-knight, 3 black-pawns, 1 black-rook, 1 white-bishop, 1 white-king, 1 white-knight, 3 white-pawns, 1
Speed: 1.0ms preprocess, 13.0ms inference, 6.0ms postprocess per image at shape (1, 3, 416, 416)
[[['', '', '', 'black-king', '', '', '', ''], ['', '', '', 'black-pawn', '', '', '', ''], ['', '', '', '', '', 'black-pawn', '', ''],
3k4/3p4/5p2/6p1/3RB1N1/1Q3P2/rP1P2n1/3K4
```

La detección para una imagen cualquiera perteneciente al conjunto test vemos que se realiza con una precisión total.



Posición de las piezas detectadas en el tablero normalizado.



Virtualización

Por último con todo listo y la línea de texto en formato fen preparada, generamos la posición del tablero con la librería chess y, con ayuda de la librería pygame creamos el tablero de forma gráfica. Para las piezas del tablero fue necesario extraer imágenes en formato png de internet.

Inicialización de pygame y establecimiento de la posición.

```
import pygame
import chess

# Inicializar pygame
pygame.init()

# Tamaño del tablero
BOARD_SIZE = 800
SQUARE_SIZE = BOARD_SIZE // 8

# Colores
LIGHT_COLOR = (240, 217, 181)
DARK_COLOR = (181, 136, 99)

# Inicializar ventana de pygame
screen = pygame.display.set_mode((BOARD_SIZE, BOARD_SIZE))
pygame.display.set_caption("Tablero de Ajedrez")

# Crear un tablero de ajedrez y cargar un FEN
fen_position = fen
board = chess.Board()
board.set_fen(fen)
```

Función para crear el tablero.

```
# Función para dibujar el tablero
def draw_board(screen, board):
    # Dibujar las casillas
    for rank in range(8):
        for file in range(8):
            color = LIGHT_COLOR if (rank + file) % 2 == 0 else DARK_COLOR
            pygame.draw.rect(screen, color, (file * SQUARE_SIZE, rank * SQUARE_SIZE, SQUARE_SIZE, SQUARE_SIZE))

    # Dibujar las piezas
    pieces = board.piece_map()
    for square, piece in pieces.items():
        # Calcular las coordenadas de la pieza
        rank = 7 - chess.square_rank(square)
        file = chess.square_file(square)

        # Determinar la carpeta según el color de la pieza
        subdir = "white" if piece.color == chess.WHITE else "black"

        # Cargar la imagen de la pieza
        piece_img = pygame.image.load(f"piezas/{subdir}/{piece.symbol().lower()}.png")
        piece_img = pygame.transform.scale(piece_img, (SQUARE_SIZE, SQUARE_SIZE))

        # Dibujar la pieza en la pantalla
        screen.blit(piece_img, (file * SQUARE_SIZE, rank * SQUARE_SIZE))
```

Pngs correspondientes a las piezas del tablero sacados de internet.



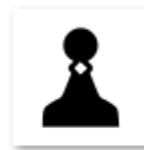
b.png



k.png



n.png



p.png



q.png



r.png



B.png



K.png



N.png



P.png



Q.png

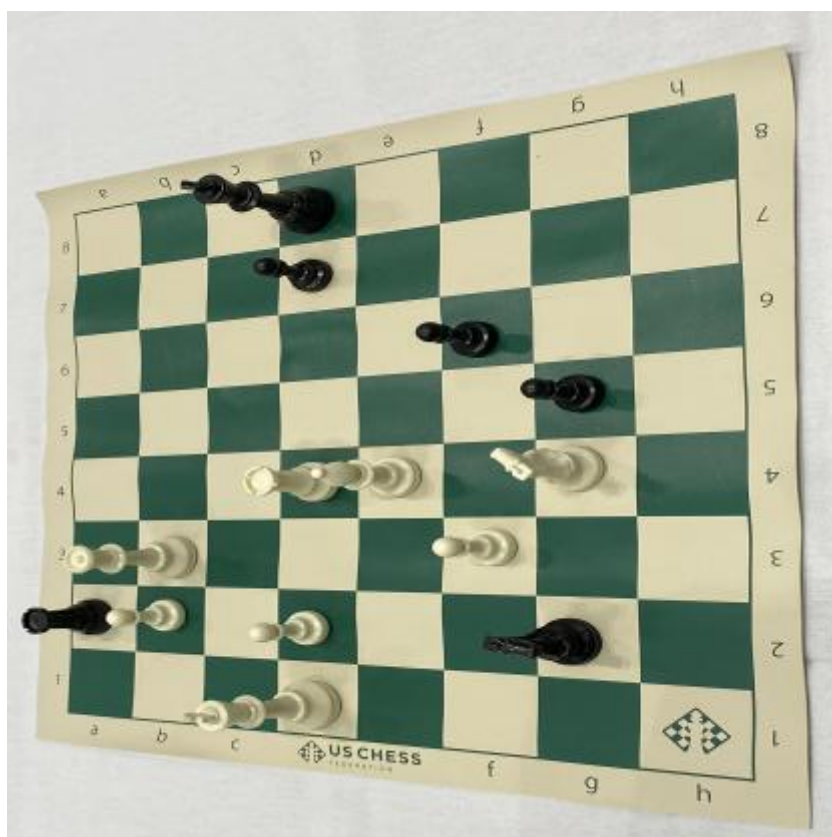


R.png

Posición final virtualizada en un tablero generado con pygames.



El resultado es un éxito total, la posición virtualizada corresponde exactamente a la de su origen en el tablero físico. Esto se puede apreciar más fácilmente comparándola con la imagen orinal del tablero rotada.



CONCLUSIÓN Y PROPUESTAS DE AMPLIACIÓN

Los objetivos de la práctica han sido cumplidos de forma muy exitosa, si se prueba el programa con otras imágenes pertenecientes al directorio test del conjunto de datos funciona muy bien, no obstante, existe un buen margen de mejora detallado en las siguientes propuestas de ampliación.

- Detección automática de esquinas: para la detección del tablero se podría decir que hago un poco de trampa puesto que, realizo de forma manual la especificación de las coordenadas de este, lo ideal es que este proceso fuese automático y mas teniendo en cuenta el siguiente punto.
- Detección en video: implementación de la práctica para la detección en video de una partida de ajedrez, idealmente en tiempo real.
- Mejora del dataset y detección: con el conjunto de datos que se ha entrenado el modelo, este funciona muy bien para imágenes del tablero desde esas perspectivas, ese tipo de tablero, esa escala y esas piezas concretas, sin embargo, para imágenes en las que se alteran un poco estas variables tiene dificultades, sería muy interesante una mayor generalización del modelo mediante el uso de técnicas de aumentación de los datos y al mismo tiempo ampliación manual del dataset.
- Mejora de la interfaz del tablero virtualizado: también sería muy interesante mejorar la interfaz gráfica del tablero virtual, ahora mismo solo muestra la posición y no se puede interactuar con él. Estaría bien que se pudiese modificar la posición haciendo jugadas, que se mostrase quien va ganando y jugadas candidatas.
- Integración en una aplicación: si se consigue todo lo anterior se podría integrar en una aplicación.