Universidad de Sevilla

Escuela Politécnica
Superior de Sevilla

Trabajo Fin de Grado en Ingeniería Electrónica Industrial

Prototipo de control brazo robótico industrial con detección de objetos basado en Deep-Learning

ANEXO 2. CÓDIGO DEL PROYECTO.

Autor: Roque Sánchez Ferrera

Tutores: Alejandro Linares Barranco y Enrique Piñero Fuentes

Fecha de Presentación: Julio 2023

En este Anexo 2 se muestra el código del proyecto desarrollado en Visual Studio Code mediante el lenguaje de programación Python.

```
import cv2
import numpy as np
import matlab.engine

# Función para calibrar la cámara
def calibrar_camara(captura):

    try:
        # Cargar los parámetros de calibración desde los archivos de
texto
        camera_matrix = np.loadtxt('calibration_parameters.txt')
        dist_coeffs = np.loadtxt('distortion_coefficients.txt')

    except FileNotFoundError:

        # Tamaño del tablero de calibración (número de esquinas internas)
        calibration_board_size = (9, 6)

        # Definir puntos en el mundo real (esquinas del tablero de
calibración)
        object_points = np.zeros((np.prod(calibration_board_size), 3),
dtype=np.float32)
        object_points[:, :2] = np.mgrid[0:calibration_board_size[0],
0:calibration_board_size[1]].T.reshape(-1, 2)

        # Almacenar puntos 3D del mundo real y puntos 2D de la imagen
para cada imagen calibrada
        object_points_list = []  # Puntos 3D en el mundo real
        image_points_list = []   # Puntos 2D en la imagen

        print("Pulse c para guardar imágenes y q cuando termine")

        while True:
            ret, frame = captura.read()  # Leer imagen de la webcam

            # Dibujar el tablero de calibración en la imagen
            found, corners = cv2.findChessboardCorners(frame,
calibration_board_size)
            if found:
                cv2.drawChessboardCorners(frame, calibration_board_size,
corners, found)

            cv2.imshow('Webcam Calibration', frame)
```

```
            key = cv2.waitKey(1)
            if key == ord('q'):
                break
            elif key == ord('c') and found:
                # Almacenar puntos 3D y 2D
                object_points_list.append(object_points)
                image_points_list.append(corners)

                print("Captured image for calibration.")

        # Calibrar la cámara y obtener los parámetros de distorsión
        ret, camera_matrix, dist_coeffs, _, _ =
cv2.calibrateCamera(object_points_list, image_points_list,
frame.shape[:2], None, None)

        # Guardar los parámetros en un archivo de texto
        np.savetxt('calibration_parameters.txt', camera_matrix)
        np.savetxt('distortion_coefficients.txt', dist_coeffs)

    return camera_matrix, dist_coeffs

def seleccionar_roi(event, x, y, flags, param):
    global inicio_x, inicio_y, fin_x, fin_y, recortando

    if event == cv2.EVENT_LBUTTONDOWN:
        inicio_x, inicio_y = x, y
        recortando = True

    elif event == cv2.EVENT_LBUTTONUP:
        fin_x, fin_y = x, y
        recortando = False
        cv2.imshow("Recorte de imagen", image)

def recortar_imagen_raton(captura):

    #Vector de coordenadas
    coordenadas = []

    global image
    _, image = captura.read()
    image = cv2.undistort(image, matriz_camara, coeficientes_distorsion)

    try:
        archivo = open('Coordenadas_primer_recorte.txt', "r")
        contenido = archivo.readlines()
```

```
        archivo.close()

        for linea in contenido:
            linea = linea.strip() # Elimina los espacios en blanco al
inicio y al final de la línea
            coordenadas.append(linea)

    except FileNotFoundError:

        print("Arrastre el ratón sobre la ventana para recortar el área
de trabajo")
        # Crear una ventana y asociar la función de retrollamada del
ratón
        cv2.namedWindow("Recorte de imagen")
        cv2.setMouseCallback("Recorte de imagen", seleccionar_roi)

        while True:
            # Mostrar la imagen y esperar la tecla 'q' para salir
            cv2.imshow("Recorte de imagen", image)
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                break

            # Actualizar el rectángulo de selección en tiempo real
            if recortando:
                frame = image.copy()
                cv2.imshow("Recorte de imagen", frame)

        #Abrir archivo txt para guardar coordenadas
        archivo = open('Coordenadas_primer_recorte.txt', "w")

        coordenadas.append(inicio_x)
        coordenadas.append(inicio_y)
        coordenadas.append(fin_x)
        coordenadas.append(fin_y)

        for i in coordenadas:
            archivo.write('{}\n'.format(i))
        #Cerramos el archivo
        archivo.close()
        cv2.destroyAllWindows()

    # Recortar la región de interés (ROI)
    roi = image[inicio_y:fin_y, inicio_x:fin_x]
```

```
    return roi, coordenadas[0], coordenadas[1], coordenadas[2],
coordenadas[3]

# Función para detectar el área de trabajo en la imagen y calcular la
relación entre píxeles y cm
def detectar_esquinas(roi):

    #Vector de coordenadas
    coordenadas = []

    try:
        archivo = open('Coordenadas_segundo_recorte.txt', "r")
        contenido = archivo.readlines()
        archivo.close()

        for linea in contenido:
            linea = linea.strip() # Elimina los espacios en blanco al
inicio y al final de la línea
            coordenadas.append(linea)

    except FileNotFoundError:

        # Convertir la imagen a escala de grises
        gris = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

        # Detectar esquinas con la función goodfeaturesToTrack
        esquinas = cv2.goodFeaturesToTrack(gris, 8, 0.01, 10)
        esquinas = np.int0(esquinas)

        # we iterate through each corner,
        # making a circle at each point that we think is a corner.
        for i in esquinas:
            x, y = i.ravel()
            cv2.circle(roi, (x, y), 3, (0,0,255), -1)
            cv2.putText(roi, 'x={},y={}'.format(x,y), (x+10, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)

        print("Pulse esc e inserte las coordenadas correspondientes")
        while(1):
            cv2.imshow('Coordenadas esquinas', roi)
            k = cv2.waitKey(30) & 0xff
            if k==27:
                break

        #Introducir coordenadas de las esquinas del área de trabajo
```

```
        punto = input("Coordenada X esquina superior izquierda: ") #Punto
1 x
        coordenadas.append(punto)
        punto = input("Coordenada Y esquina superior izquierda: ") #Punto
1 y
        coordenadas.append(punto)
        punto = input("Coordenada X esquina superior derecha: ") #Punto 2
x
        coordenadas.append(punto)
        punto = input("Coordenada Y esquina superior derecha: ") #Punto 2
y
        coordenadas.append(punto)
        punto = input("Coordenada X esquina inferior izquierda: ") #Punto
3 x
        coordenadas.append(punto)
        punto = input("Coordenada Y esquina inferior izquierda: ") #Punto
3 y
        coordenadas.append(punto)
        punto = input("Coordenada X esquina inferior derecha: ") #Punto 4
x
        coordenadas.append(punto)
        punto = input("Coordenada Y esquina inferior derecha: ") #Punto 4
y
        coordenadas.append(punto)

        #Abrir archivo txt para guardar coordenadas
        archivo = open('Coordenadas_segundo_recorte.txt', "w")
        for i in coordenadas:
            archivo.write('{}\n'.format(i))
        #Cerramos el archivo
        archivo.close()

        # Liberar los recursos y cerrar las ventanas
        """captura.release()
        cv2.destroyAllWindows()"""

    #Cálculo de la relación entre píxeles y cm
    distancia_real_x = 18.8 #Configurar distancia real que existe en
centímetros eje x
    distancia_real_y = 18.8 #Configurar distancia real que existe en
centímetros eje y

    distancia_pixel_x1 = float(coordenadas[2])-float(coordenadas[0])
    distancia_pixel_x2 = float(coordenadas[6])-float(coordenadas[4])

    distancia_pixel_y1 = float(coordenadas[5])-float(coordenadas[1])
```

```
    distancia_pixel_y2 = float(coordenadas[7])-float(coordenadas[3])

    relacion_x1 = distancia_pixel_x1/distancia_real_x
    relacion_x2 = distancia_pixel_x2/distancia_real_x

    relacion_y1 = distancia_pixel_y1/distancia_real_y
    relacion_y2 = distancia_pixel_y2/distancia_real_y

    relacion_pixelcm =
(relacion_x1+relacion_x2+relacion_y1+relacion_y2)/4

    print('Un cm en el mundo real equivale a {:.2f}
píxeles'.format(relacion_pixelcm))

    return coordenadas[0], coordenadas[1], coordenadas[2],
coordenadas[5], relacion_pixelcm

# Función para recortar la imagen capturada por la webcam y sacar las
coordenadas x e y
def recortar_areatrabajo_coordenadas(imagen, punto_1_x, punto_1_y,
punto_2_x, punto_3_y):
     # Recortar área de trabajo
     areatrabajo =  imagen[punto_1_y:punto_3_y, punto_1_x:punto_2_x,]

     return areatrabajo

# Función para recortar la imagen capturada por la webcam y sacar el
ángulo
def recortar_areatrabajo_angulo(imagen, punto_1_x, punto_1_y, punto_2_x,
punto_3_y):
     # Recortar área de trabajo
     areatrabajo =  imagen[punto_1_y+5:punto_3_y-5,
punto_1_x+5:punto_2_x-5]

     return areatrabajo

# Función para detectar objetos en la imagen utilizando YOLOv3
def detectar_objetos(captura):

    while True:
        objeto_detectado = None #En caso de que no se detecte ningún
objeto devolverá none
        (H, W) = (None, None)
        # Leer fotograma
        (ret, imagen1) = captura.read()
        (ret, imagen2) = captura.read()
```

```
        #Eliminar distorsión
        imagen_corregido1 = cv2.undistort(imagen1, matriz_camara,
coeficientes_distorsion)
        imagen_corregido2 = cv2.undistort(imagen2, matriz_camara,
coeficientes_distorsion)

        #Recortar áreas de trabajo
        frame = recortar_areatrabajo_coordenadas(imagen_corregido1,
esquina_1_x, esquina_1_y, esquina_2_x, esquina_3_y)
        frame_angulo = recortar_areatrabajo_angulo(imagen_corregido2,
esquina_1_x, esquina_1_y, esquina_2_x, esquina_3_y)

        if W is None or H is None:
            (H,W) = frame.shape[:2]

        blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB
= True, crop = False)
        net.setInput(blob)
        layersOutputs = net.forward(outputLayer)

        boxes = []
        confidences = []
        classIDs = []

        for output in layersOutputs:
            for detection in output:
                scores = detection[5:]
                classID = np.argmax(scores)
                confidence = scores[classID]
                if confidence > confidenceThreshold:
                    box = detection[0:4] * np.array([W, H, W, H])
                    (centerX, centerY,  width, height) =
box.astype('int')

                    x = int(centerX - (width/2))
                    y = int(centerY - (height/2))

                    boxes.append([x, y, int(width), int(height)])
                    confidences.append(float(confidence))
                    classIDs.append(classID)

        detectionNMS = cv2.dnn.NMSBoxes(boxes, confidences,
confidenceThreshold, NMSThreshold)

        if(len(detectionNMS) > 0):
```

```
            gray = cv2.cvtColor(frame_angulo, cv2.COLOR_BGR2GRAY)
            thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)[1]

            # Find contours, find rotated rectangle, obtain four
verticies, and draw
            cnts = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
            cnts = cnts[0] if len(cnts) == 2 else cnts[1]
            rect = cv2.minAreaRect(cnts[0])
            (x, y), (ancho, alto), angulo = cv2.minAreaRect(cnts[0])

            if ancho>10 and alto >10:
                if ancho < alto:
                    angulo += 90

                box = np.int0(cv2.boxPoints(rect))
                #color = [int(c) for c in COLORS[classIDs[i]]]
                cv2.drawContours(frame_angulo, [box], 0, 0, 3) # OR

                for i in detectionNMS.flatten():

                    (x, y) = (boxes[i][0], boxes[i][1])
                    (w, h) = (boxes[i][2], boxes[i][3])

                    objeto = labels[classIDs[i]]
                    print ("Objeto detectado:", objeto)

                    x1 = int(x + w/2) #Coordenadas en píxeles centro del
objeto
                    y1 = int(y + h/2)

                    x_real = x1/relacion_pixelcm
                    y_real = y1/relacion_pixelcm

                    color = [int(c) for c in COLORS[classIDs[i]]]
                    cv2.rectangle(frame, (x, y), (x + w, y + h), color,
2) #Rectángulo alrededor del objeto

                objeto_detectado = {
                "objeto": objeto,
                "probabilidad": confidence,
                "coordenadas": (x_real, y_real),
                "angulo": angulo
                }
```

```
        cv2.imshow('Webcam', frame)
        cv2.imshow('Webcam2', frame_angulo)
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break

    return objeto_detectado

# Cargar los archivos de configuración y pesos de YOLOv3
modelConfiguration = 'Yolo_train\yolov3.cfg'
modelWeights = 'Yolo_train\yolov3.weights'
labelsPath = 'Yolo_train\coco.names'
confidenceThreshold = 0.5
NMSThreshold = 0.3
labels = open(labelsPath).read().strip().split('\n')
np.random.seed(10)
COLORS = np.random.randint(0, 255, size=(len(labels), 3), dtype="uint8")
net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
outputLayer = net.getLayerNames()
outputLayer = [outputLayer[i - 1] for i in net.getUnconnectedOutLayers()]

# Capturar imágenes de la webcam para la calibración
captura = cv2.VideoCapture(0)  # 0 para la primera webcam
captura.set(cv2.CAP_PROP_FRAME_WIDTH, 640)  # Ancho de la imagen
capturada
captura.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # Alto de la imagen
capturada

# Calibrar la cámara
matriz_camara, coeficientes_distorsion = calibrar_camara(captura)

# Variables globales
inicio_x, inicio_y = -1, -1
fin_x, fin_y = -1, -1
recortando = False

# Detectar el área de trabajo
roi, inicio_x, inicio_y, fin_x, fin_y = recortar_imagen_raton(captura)
punto_1_x, punto_1_y, punto_2_x, punto_3_y, relacion_pixelcm =
detectar_esquinas(roi)

esquina_1_x = int(punto_1_x) + int(inicio_x)
esquina_1_y = int(punto_1_y) + int(inicio_y)
esquina_2_x = int(punto_2_x) + int(inicio_x)
esquina_3_y = int(punto_3_y) + int(inicio_y)
```

```
# Inicializar motor de matlab
eng = matlab.engine.start_matlab()

# Inicializar scorbot. Ejecutar antes ScorbotServer.exe
eng.ScorInit()
eng.ScorHome()

# Configurar velocidad del robot
eng.ScorSetSpeed(100)

# Definir posicion del robot en la esquiana superior izquierda. Punto de
referencia
X_ref = 400.000
Y_ref = 100.000
Z_ref = 0.000
Z_ref_naranja = 20.000 #Debido a la altura de la naranja
Pitch_ref = -1.500
Roll_ref = 0.000

# Llamar a la función para capturar video, detectar y recortar el área de
trabajo
while True:

    objeto_detectado = detectar_objetos(captura)

    if objeto_detectado is not None:

        objeto = objeto_detectado["objeto"]
        objeto = str(objeto)

        if objeto=='mouse' or objeto=='orange' or objeto=='cell phone' or
objeto=='remote':

            (x, y) = objeto_detectado["coordenadas"]
            x = float(x)
            y = float(y)

            angulo = objeto_detectado["angulo"]
            angulo = float(angulo)
            angulo = matlab.double(angulo)
            angulo = eng.deg2rad(angulo)

            # Configurar relación del robot (coger distancia real para el
robot)
```

```
            relacion_robotcm = 9.574 #180/18.8 (distancia del robot de
esquina a esquina/distancia real)
            X = X_ref - (relacion_robotcm*y)
            Y = Y_ref - (relacion_robotcm*x)
            Pitch = -1.5
            Roll = angulo - np.pi/2

            if objeto=='orange':
                Z = Z_ref_naranja + ((10/18.8)*y) + 25 #25 es la altura
de seguridad
            elif objeto=='cell phone' or objeto=='remote' or
objeto=='mouse':
                Z = Z_ref + ((5/18.8)*y) + 25

            print('Moviendo robot a (x={:.2f}cm, y={:.2f}cm).
Ángulo:{:.2f}'.format(x,y,angulo))

            posicion = matlab.double([X,Y,Z,Pitch,Roll])

            # Open Gripper
            eng.ScorSetGripper('Open')
            eng.ScorWaitForMove()

            eng.ScorSetXYZPR(posicion)
            eng.ScorWaitForMove()

            Z = Z - 25 #Bajamos a coger el objeto

            posicion = matlab.double([X,Y,Z,Pitch,Roll])

            eng.ScorSetXYZPR(posicion)
            eng.ScorWaitForMove()

            # Close Gripper
            eng.ScorSetGripper('Close')
            eng.ScorWaitForMove()

            Z = Z + 50 #Volvemos a subir

            posicion = matlab.double([X,Y,Z,Pitch,Roll])

            eng.ScorSetXYZPR(posicion)
            eng.ScorWaitForMove()

            # Clasificar los objetos en sus correspondientes depóitos
            if objeto=='mouse':
```

```
                X = 175.00
                Y = 250.00
                Z = 80.00 + 50
                Pitch = -1.5
                Roll = 0
            elif objeto=='orange':
                X = 350.00
                Y = 250.00
                Z = 100.00 + 50
                Pitch = -1.5
                Roll = 0
            elif objeto=='cell phone':
                X = 350.00
                Y = -220.00
                Z = 100.00 + 50
                Pitch = -1.5
                Roll = np.pi/4
            elif objeto=='remote':
                X = 235.00
                Y = -220.00
                Z = 100.00 + 50
                Pitch = -1.5
                Roll = np.pi/4

            posicion = matlab.double([X,Y,Z,Pitch,Roll])

            eng.ScorSetXYZPR(posicion)
            eng.ScorWaitForMove()

            Z = Z - 75 #Bajamos a dejar el objeto

            posicion = matlab.double([X,Y,Z,Pitch,Roll])

            eng.ScorSetXYZPR(posicion)
            eng.ScorWaitForMove()

            # Open Gripper
            eng.ScorSetGripper('Open')
            eng.ScorWaitForMove()

            eng.ScorGoHome()
            eng.ScorWaitForMove()

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
```

```
# Safe shutdown
eng.ScorSafeShutdown()
eng.quit()
captura.release()
cv2.destroyAllWindows
```