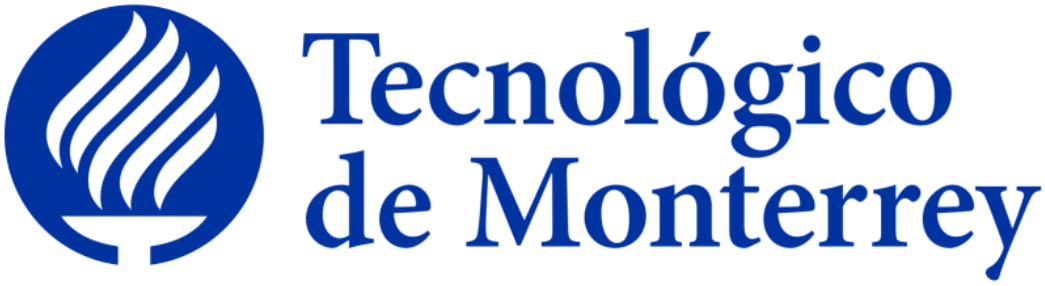
**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Campus Monterrey**




Implementación de robótica inteligente

TE3002B, Grupo 501

# Visual Control of Duck Detection and Tracking System

Elaborado por:

Emilio Rizo De la Mora | A01721612

Jose Pablo Cedano Serna | A00832019

Luis Antonio Zermeño De Gorordo | A01781835

Omar Flores Sanchéz | A01383530

Luis Mario Lozoya Chairez | A00833364

Jorge Axel Castruita Bretado | A00832843

Rodrigo Escandón López Guerrero | A01704287

Dr. Luis Alberto Muñoz

## Secciones:

## Nuestro entregable se divide en diferentes códigos.

1) Código para la detección de los patos y su tracking
2) 2 códigos para la creación del modelo del pato. El primer código para analizar y generar los datos del modelo y el segundo código para abrir estos datos.
3) Código para la recreación del camarman movement.

## Código 1

## Introducción

Este proyecto se centra en la detección y seguimiento de patos en un video utilizando la red YOLOv5 para la detección de objetos y técnicas de calibración de cámara para calcular la distancia de los patos detectados desde la cámara. Se emplea Python con diferentes además de la implementación del modelo YOLOv5 preentrenado para identificar y rastrear los patos en tiempo real. La calibración de la cámara se realiza a partir de una imagen de referencia que contiene un patrón conocido, en este caso siendo las lozas que aparecen en el piso del video.

## Características

1. **Detección de Bordes e Intersecciones:** Utiliza técnicas de procesamiento de imágenes para detectar bordes y calcular intersecciones en una imagen de referencia.
2. **Calibración de Cámara:** Estima los parámetros intrínsecos de la cámara (matriz de la cámara y coeficientes de distorsión) utilizando puntos de referencia de una imagen.
3. **Detección y Seguimiento de Patos:** Emplea un modelo YOLOv5 para detectar patos en un video y calcula la distancia de los patos a la cámara basándose en la geometría de los mismos.
4. **Visualización en Tiempo Real:** Muestra los resultados de la detección y el seguimiento en tiempo real, resaltando los patos detectados y mostrando la distancia calculada.

Python

```python
import cv2
import torch
import numpy as np
```

```
import os
import json

# Función para detectar los bordes y encontrar los puntos de
intersección
def find_intersections(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180,
threshold=100, minLineLength=100, maxLineGap=10)
    intersections = []
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(img, (x1, y1), (x2, y2), (0, 255, 255),
2)  # Dibuja las líneas en amarillo

        for i in range(len(lines)):
            for j in range(i + 1, len(lines)):
                line1, line2 = lines[i][0], lines[j][0]
                intersection = compute_intersection(line1,
line2)
                if intersection:
                    intersections.append(intersection)
                    cv2.circle(img, intersection, 5, (0, 0,
255), -1)  # Dibuja los puntos de intersección en rojo
    return img, intersections

# Función para calcular la intersección de dos líneas
def compute_intersection(line1, line2):
    x1, y1, x2, y2 = line1
    x3, y3, x4, y4 = line2
```

```
    denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
    if denom == 0:
        return None
    px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4
- y3 * x4)) / denom
    py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4
- y3 * x4)) / denom
    return int(px), int(py)

# Función para calcular la distancia utilizando tamaño
def calculate_distance(apparent_size, real_size_width,
real_size_height, focal_length_width, focal_length_height):
    if apparent_size[0] == 0 or apparent_size[1] == 0:
        return float('inf')  # Evita divisiones por cero
    distance_width = (real_size_width * focal_length_width) /
apparent_size[0]
    distance_height = (real_size_height * focal_length_height) /
apparent_size[1]
    distance = (distance_width + distance_height) / 2
    distance /= .37  # Convertir mm a cm
    return distance

# Función para calcular los valores focales
def calculate_focal_lengths(mtx, pixel_size):
    focal_length_width = mtx[0][0] * pixel_size
    focal_length_height = mtx[1][1] * pixel_size
    return focal_length_width, focal_length_height

# Función para ajustar el tamaño del pixel y calcular los
valores focales
# al valor focal le damos un rangos de tolerancia dentro de la
iteracion de su calculo para encontrar el valor optimo
```

```
def adjust_pixel_size_to_focal_range(mtx,
target_focal_range=(90, 200), max_iterations=100):
    tolerance = 0.000025  # Incremento/decremento inicial del
tamaño del pixel
    pixel_size = 0.00001  # Tamaño inicial del pixel en cm
    iteration_count = 0
    adjusted = False

    while iteration_count < max_iterations:
        focal_length_width, focal_length_height =
calculate_focal_lengths(mtx, pixel_size)
        print(f"Current Focal Width: {focal_length_width},
Current Focal Height: {focal_length_height}")

        if target_focal_range[0] <= focal_length_width <=
target_focal_range[1] and target_focal_range[0] <=
focal_length_height <= target_focal_range[1]:
            adjusted = True
            break

        # Ajustar el tamaño del pixel en función de si está por
encima o por debajo del rango deseado
        if focal_length_width > target_focal_range[1] or
focal_length_height > target_focal_range[1]:
            pixel_size -= tolerance  # Disminuir si los valores
focales son demasiado altos
        else:
            pixel_size += tolerance  # Aumentar si los valores
focales son demasiado bajos

        iteration_count += 1
```

```
    if not adjusted:
        print("No se pudo ajustar el tamaño del pixel para
obtener valores focales dentro del rango especificado.")

    return pixel_size, focal_length_width, focal_length_height

# Carga y calibración de la imagen
def calibrate_camera_from_image(image_path):
    img = cv2.imread(image_path)
    assert img is not None, "Error: la imagen no pudo ser
cargada."
    height, width = img.shape[:2]
    aspect_ratio = width / height
    new_width = 500
    new_height = int(new_width / aspect_ratio)
    img = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_AREA)

    img_with_intersections, intersections =
find_intersections(img)

    img_name = os.path.basename(image_path)
    puntos_ref = {img_name: intersections}
    with open('puntos_referencia.json', 'w') as fp:
        json.dump(puntos_ref, fp)

    puntos_imagen = []
    puntos_objeto = []

    loza_ancho = 30  # Ancho en centímetros
    loza_alto = 30  # Alto en centímetros
```

```
    for img_name, puntos in puntos_ref.items():
        for punto in puntos:
            puntos_imagen.append([punto])
            x, y = punto
            puntos_objeto.append([(float(x / img.shape[1]) *
loza_ancho, float(y / img.shape[0]) * loza_alto, 0.0)])

    puntos_imagen = np.array(puntos_imagen, dtype=np.float32)
    puntos_objeto_nested = [tuple(punto) for punto in
puntos_objeto]
    puntos_objeto = np.array(puntos_objeto_nested,
dtype=np.float32)

    ret, mtx, dist, rvecs, tvecs =
cv2.calibrateCamera([puntos_objeto], [puntos_imagen],
(img.shape[1], img.shape[0]), None, None)

    print("Matriz de la cámara (parámetros intrínsecos):")
    print(mtx)
    print("\nCoeficientes de distorsión:")
    print(dist)
    print("\nVectores de rotación:")
    print(rvecs)
    print("\nVectores de traslación:")
    print(tvecs)

    # Ajustar el tamaño del pixel para obtener valores focales
dentro del rango especificado
    pixel_size, focal_length_width, focal_length_height =
adjust_pixel_size_to_focal_range(mtx)
    print("\nAdjusted Pixel Size (cm):")
    print(pixel_size)
```

```
    print("\nAdjusted Focal Width (cm):")
    print(focal_length_width)
    print("\nAdjusted Focal Height (cm):")
    print(focal_length_height)

    # Mostrar imagen con intersecciones detectadas
    cv2.imshow('Intersections', img_with_intersections)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar la imagen calibrada
    img = cv2.imread(image_path, 1)
    img = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_AREA)
    h, w = img.shape[:2]
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist,
(w, h), 1, (w, h))
    dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
    cv2.imwrite('calibresult.png', dst)

    # Retornar los parámetros de calibración
    return mtx, dist, pixel_size, focal_length_width,
focal_length_height

# Función para detección y tracking de patos
def detect_and_track_ducks(video_path, mtx, dist, pixel_size,
focal_length_width, focal_length_height, delay):

    duck_height_cm = 8.0
    duck_width_cm = 12.0
```

```
    # Cargar el modelo YOLOv5 desde el archivo entrenado
    model_path =
r'C:\Users\rodri\yolov5\runs\train\exp5\weights\best.pt'
    model = torch.hub.load('ultralytics/yolov5', 'custom',
path=model_path)

    cap = cv2.VideoCapture(video_path)
    assert cap.isOpened(), "Error: No se puede abrir el video."

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

        # Detección de patos utilizando YOLOv5
        results = model(frame)

        # Extraer coordenadas y etiquetas de los resultados
        detections = results.xyxy[0].cpu().numpy()

        # Encontrar el pato con la mejor confianza de detección
        best_conf = 0
        best_box = None
        for *box, conf, cls in detections:
            if conf > best_conf:
                best_conf = conf
                best_box = box

        # Enmarcar a los patos detectados
        for *box, conf, cls in detections:
            x1, y1, x2, y2 = map(int, box)
```

```
            label = f'{model.names[int(cls)]} {conf * 100:.1f}%'
            color = (0, 255, 0)  # Verde para todos los patos
            if box == best_box:
                color = (0, 0, 255)  # Rojo para el pato con
mayor % de confianza
                # Calcular la distancia utilizando la altura de
la caja delimitadora
                apparent_width = x2 - x1
                apparent_height = y2 - y1
                distance_cm = calculate_distance(
                    (apparent_width, apparent_height),
                    duck_width_cm, duck_height_cm,
                    focal_length_width, focal_length_height
                )
                #print(f'Apparent Width: {apparent_width},
Apparent Height: {apparent_height}, Distance: {distance_cm} cm')
                distance_label = f'Distancia: {distance_cm:.1f}
cm'
                cv2.putText(frame, distance_label, (x1, y1 -
30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
            cv2.putText(frame, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

        # Mostrar el frame procesado
        cv2.imshow('Ducks Detection', frame)

        if cv2.waitKey(delay) & 0xFF == ord(' '):
            break

    cap.release()
```

```
    cv2.destroyAllWindows()

if __name__ == "__main__":
    image_path = 'C:/Users/rodri/MR5 TrackingPatos/PatronLoza/Captura de pantalla 2024-05-15 222749.png'
    video_path = r'C:\Users\rodri\MR5 TrackingPatos\DuckVideo.mp4'

    # Calibrar la cámara
    mtx, dist, pixel_size, focal_length_width, focal_length_height = calibrate_camera_from_image(image_path)

    # Detectar y trackear patos, ajustando el delay por que el video se procesaba muy lento
    detect_and_track_ducks(video_path, mtx, dist, pixel_size, focal_length_width, focal_length_height, delay=5)
```

## Código 2

## Introducción

Este codigo amplía la funcionalidad del sistema de detección y seguimiento de patos, integrando la generación de modelos 2D y 3D de los patos detectados. Utilizando el modelo YOLOv5 para la detección y técnicas avanzadas de procesamiento de imágenes y computación 3D, se captura la imagen del pato con mayor % de confianza de detección y se genera un modelo tridimensional del mismo.

## Características

1. **Detección y Seguimiento Mejorados:** Selección del pato con la mayor confianza de detección y extracción de la imagen del objeto detectado.
2. **Generación de Modelos 2D:** Guarda la imagen 2D del pato detectado con mayor confianza.
3. **Generación de Modelos 3D:** Utiliza la biblioteca Open3D para generar un modelo 3D de la imagen 2D del pato detectado.

Python

```python
import cv2
import torch
import numpy as np
import os
import json
import open3d as o3d

# Función para detectar los bordes y encontrar los puntos de intersección
def find_intersections(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=100, minLineLength=100, maxLineGap=10)
    intersections = []
    if lines is not None:
        for line in lines:
```

```
            for x1, y1, x2, y2 in line:
                cv2.line(img, (x1, y1), (x2, y2), (0, 255, 255),
2)

        for i in range(len(lines)):
            for j in range(i + 1, len(lines)):
                line1, line2 = lines[i][0], lines[j][0]
                intersection = compute_intersection(line1,
line2)
                if intersection:
                    intersections.append(intersection)
                    cv2.circle(img, intersection, 5, (0, 0,
255), -1)
    return img, intersections

# Función para calcular la intersección de dos líneas
def compute_intersection(line1, line2):
    x1, y1, x2, y2 = line1
    x3, y3, x4, y4 = line2
    denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
    if denom == 0:
        return None
    px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4
- y3 * x4)) / denom
    py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4
- y3 * x4)) / denom
    return int(px), int(py)

# Función para calcular la distancia utilizando tamaño
def calculate_distance(apparent_size, real_size_width,
real_size_height, focal_length_width, focal_length_height):
    if apparent_size[0] == 0 or apparent_size[1] == 0:
```

```
        return float('inf')
    distance_width = (real_size_width * focal_length_width) /
apparent_size[0]
    distance_height = (real_size_height * focal_length_height) /
apparent_size[1]
    distance = (distance_width + distance_height) / 2
    distance /= .37
    return distance

# Función para calcular los valores focales
def calculate_focal_lengths(mtx, pixel_size):
    focal_length_width = mtx[0][0] * pixel_size
    focal_length_height = mtx[1][1] * pixel_size
    return focal_length_width, focal_length_height

# Función para ajustar el tamaño del pixel y calcular los
valores focales
def adjust_pixel_size_to_focal_range(mtx,
target_focal_range=(90, 200), max_iterations=100):
    tolerance = 0.000025
    pixel_size = 0.00001
    iteration_count = 0
    adjusted = False

    while iteration_count < max_iterations:
        focal_length_width, focal_length_height =
calculate_focal_lengths(mtx, pixel_size)
        print(f"Current Focal Width: {focal_length_width},
Current Focal Height: {focal_length_height}")
```

```
        if target_focal_range[0] <= focal_length_width <=
target_focal_range[1] and target_focal_range[0] <=
focal_length_height <= target_focal_range[1]:
            adjusted = True
            break

        if focal_length_width > target_focal_range[1] or
focal_length_height > target_focal_range[1]:
            pixel_size -= tolerance
        else:
            pixel_size += tolerance

        iteration_count += 1

    if not adjusted:
        print("No se pudo ajustar el tamaño del pixel para
obtener valores focales dentro del rango especificado.")

    return pixel_size, focal_length_width, focal_length_height

# Carga y calibración de la imagen
def calibrate_camera_from_image(image_path):
    img = cv2.imread(image_path)
    assert img is not None, "Error: la imagen no pudo ser
cargada."
    height, width = img.shape[:2]
    aspect_ratio = width / height
    new_width = 500
    new_height = int(new_width / aspect_ratio)
    img = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_AREA)
```

```
    img_with_intersections, intersections =
find_intersections(img)

    img_name = os.path.basename(image_path)
    puntos_ref = {img_name: intersections}
    with open('puntos_referencia.json', 'w') as fp:
        json.dump(puntos_ref, fp)

    puntos_imagen = []
    puntos_objeto = []

    loza_ancho = 30
    loza_alto = 30

    for img_name, puntos in puntos_ref.items():
        for punto in puntos:
            puntos_imagen.append([punto])
            x, y = punto
            puntos_objeto.append([(float(x / img.shape[1]) *
loza_ancho, float(y / img.shape[0]) * loza_alto, 0.0)])

    puntos_imagen = np.array(puntos_imagen, dtype=np.float32)
    puntos_objeto_nested = [tuple(punto) for punto in
puntos_objeto]
    puntos_objeto = np.array(puntos_objeto_nested,
dtype=np.float32)

    ret, mtx, dist, rvecs, tvecs =
cv2.calibrateCamera([puntos_objeto], [puntos_imagen],
(img.shape[1], img.shape[0]), None, None)

    print("Matriz de la cámara (parámetros intrínsecos):")
```

```
    print(mtx)
    print("\nCoeficientes de distorsión:")
    print(dist)
    print("\nVectores de rotación:")
    print(rvecs)
    print("\nVectores de traslación:")
    print(tvecs)

    pixel_size, focal_length_width, focal_length_height =
adjust_pixel_size_to_focal_range(mtx)
    print("\nAdjusted Pixel Size (cm):")
    print(pixel_size)
    print("\nAdjusted Focal Width (cm):")
    print(focal_length_width)
    print("\nAdjusted Focal Height (cm):")
    print(focal_length_height)

    cv2.imshow('Intersections', img_with_intersections)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    img = cv2.imread(image_path, 1)
    img = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_AREA)
    h, w = img.shape[:2]
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist,
(w, h), 1, (w, h))
    dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
    cv2.imwrite('calibresult.png', dst)
```

```
    return mtx, dist, pixel_size, focal_length_width,
focal_length_height

def detect_and_track_ducks(video_path, mtx, dist, pixel_size,
focal_length_width, focal_length_height, delay):
    duck_height_cm = 8.0
    duck_width_cm = 12.0

    model_path =
r'C:\Users\rodri\yolov5\runs\train\exp5\weights\best.pt'
    model = torch.hub.load('ultralytics/yolov5', 'custom',
path=model_path)

    cap = cv2.VideoCapture(video_path)
    assert cap.isOpened(), "Error: No se puede abrir el video."

    best_conf = 0
    best_box = None
    best_frame = None

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

        results = model(frame)
        detections = results.xyxy[0].cpu().numpy()

        for *box, conf, cls in detections:
            if conf > best_conf:
                best_conf = conf
                best_box = box
```

```
                best_frame = frame.copy()

            x1, y1, x2, y2 = map(int, box)
            label = f'{model.names[int(cls)]} {conf * 100:.1f}%'
            color = (0, 255, 0)
            if box == best_box:
                color = (0, 0, 255)
                apparent_width = x2 - x1
                apparent_height = y2 - y1
                distance_cm = calculate_distance(
                    (apparent_width, apparent_height),
                    duck_width_cm, duck_height_cm,
                    focal_length_width, focal_length_height
                )
                distance_label = f'Distancia: {distance_cm:.1f}
cm'
                cv2.putText(frame, distance_label, (x1, y1 -
30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
            cv2.putText(frame, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

        cv2.imshow('Ducks Detection', frame)

        if cv2.waitKey(delay) & 0xFF == ord(' '):
            break

    cap.release()
    cv2.destroyAllWindows()

    if best_frame is not None and best_box is not None:
```

```
        x1, y1, x2, y2 = map(int, best_box)
        object_image = best_frame[y1:y2, x1:x2]
        cv2.imwrite('best_detected_object.png', object_image)
        return object_image

    return None

def generate_2d_model(object_image):
    if object_image is not None:
        cv2.imwrite('object_2d_model.png', object_image)
        return object_image
    return None

def generate_3d_model(object_image):
    if object_image is not None:
        gray = cv2.cvtColor(object_image, cv2.COLOR_BGR2GRAY)
        depth = cv2.Canny(gray, 50, 150)

        points = []
        for y in range(depth.shape[0]):
            for x in range(depth.shape[1]):
                if depth[y, x] > 0:
                    points.append([x, y, depth[y, x]])

        points = np.array(points)
        point_cloud = o3d.geometry.PointCloud()
        point_cloud.points = o3d.utility.Vector3dVector(points)
        o3d.io.write_point_cloud("object_3d_model.ply",
point_cloud)
        return point_cloud
    return None
```

```
if __name__ == "__main__":
    image_path = 'C:/Users/rodri/MR5
TrackingPatos/PatronLoza/Captura de pantalla 2024-05-15
222749.png'
    video_path = r'C:\Users\rodri\MR5
TrackingPatos\DuckVideo.mp4'

    mtx, dist, pixel_size, focal_length_width,
focal_length_height = calibrate_camera_from_image(image_path)
    object_image = detect_and_track_ducks(video_path, mtx, dist,
pixel_size, focal_length_width, focal_length_height, delay=5)

    if object_image is not None:
        generate_2d_model(object_image)
        generate_3d_model(object_image)
```

## Código 3

## Introducción

Este código complementa la funcionalidad de modelado 3D de patos. El código anterior se basaba en obtener los datos para poder generar el modelo mientras que este tiene como objetivo la capacidad de cargar y visualizar el modelo 3D generado. Utilizando la biblioteca Open3D, se carga el archivo de nubes de puntos (.ply) del modelo 3D y se visualiza en una ventana interactiva.

## Características

1. **Carga de Modelos 3D:** Carga archivos de nubes de puntos en formato .ply.
2. **Visualización Interactiva:** Utiliza Open3D para mostrar el modelo 3D en una ventana gráfica interactiva.

Python

```python
import open3d as o3d

def load_and_visualize_ply(file_path):
    # Cargar el archivo .ply
    pcd = o3d.io.read_point_cloud(file_path)

    # Verificar si el archivo se ha cargado correctamente
    if pcd.is_empty():
        print(f"Error: No se pudo cargar el archivo
{file_path}")
        return

    # Visualizar el modelo
    o3d.visualization.draw_geometries([pcd],
window_name="Visualizador Open3D",
                                      width=800, height=600,
left=50, top=50,
                                      point_show_normal=False,
mesh_show_wireframe=False, mesh_show_back_face=False)
```

```
if __name__ == "__main__":
    # Ruta al archivo .ply
    file_path = "object_3d_model.ply"

    # Cargar y visualizar el archivo .ply
    load_and_visualize_ply(file_path)

#stereovision opencv
#ORB
#Colmap
#ejecutable meshroom
```

## Código 4

## Introducción

Este proyecto finaliza la funcionalidad del sistema de detección y modelado de patos, añadiendo el rastreo de la trayectoria de la cámara a partir de un video. Utilizando técnicas de visión por computadora como ORB y BFMatcher, se calcula la trayectoria de la cámara en un espacio tridimensional.

## Características

1. **Rastreo de la Trayectoria de la Cámara:** Utiliza características ORB y coincidencias BFMatcher para rastrear la posición de la cámara a lo largo del tiempo.
2. **Visualización en 3D de la Trayectoria:** Muestra la trayectoria de la cámara en una gráfica 3D en vivo utilizando Matplotlib.

Python

```python
import numpy as np
import cv2
import os
import json
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Función para detectar los bordes y encontrar los puntos de intersección
def find_intersections(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=100, minLineLength=100, maxLineGap=10)
    intersections = []
    if lines is not None:
        for line in lines:
```

```
            for x1, y1, x2, y2 in line:
                cv2.line(img, (x1, y1), (x2, y2), (0, 255, 255),
2)

        for i in range(len(lines)):
            for j in range(i + 1, len(lines)):
                line1, line2 = lines[i][0], lines[j][0]
                intersection = compute_intersection(line1,
line2)
                if intersection:
                    intersections.append(intersection)
                    cv2.circle(img, intersection, 5, (0, 0,
255), -1)
    return img, intersections

# Función para calcular la intersección de dos líneas
def compute_intersection(line1, line2):
    x1, y1, x2, y2 = line1
    x3, y3, x4, y4 = line2
    denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
    if denom == 0:
        return None
    px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4
- y3 * x4)) / denom
    py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4
- y3 * x4)) / denom
    return int(px), int(py)

# Carga y calibración de la cámara desde una imagen
def calibrate_camera_from_image(image_path):
    img = cv2.imread(image_path)
```

```
    assert img is not None, "Error: la imagen no pudo ser
cargada."
    height, width = img.shape[:2]
    aspect_ratio = width / height
    new_width = 500
    new_height = int(new_width / aspect_ratio)
    img = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_AREA)

    img_with_intersections, intersections =
find_intersections(img)

    # Almacenamiento de las intersecciones detectadas (para uso
futuro)
    img_name = os.path.basename(image_path)
    puntos_ref = {img_name: intersections}
    with open('puntos_referencia.json', 'w') as fp:
        json.dump(puntos_ref, fp)

    puntos_imagen = []
    puntos_objeto = []
    credencial_ancho = 30
    credencial_alto = 30

    for img_name, puntos in puntos_ref.items():
        for punto in puntos:
            puntos_imagen.append([punto])
            x, y = punto
            puntos_objeto.append([(float(x / img.shape[1]) *
credencial_ancho, float(y / img.shape[0]) * credencial_alto,
0.0)])
```

```
    puntos_imagen = np.array(puntos_imagen, dtype=np.float32)
    puntos_objeto_nested = [tuple(punto) for punto in
puntos_objeto]
    puntos_objeto = np.array(puntos_objeto_nested,
dtype=np.float32)

    ret, mtx, dist, rvecs, tvecs =
cv2.calibrateCamera([puntos_objeto], [puntos_imagen],
(img.shape[1], img.shape[0]), None, None)

    print("Matriz de la cámara (parámetros intrínsecos):")
    print(mtx)
    print("\nCoeficientes de distorsión:")
    print(dist)
    print("\nVectores de rotación:")
    print(rvecs)
    print("\nVectores de traslación:")
    print(tvecs)

    # Calcular la longitud focal en cm
    tamano_pixel_cm = 0.000012  # Tamaño del pixel en cm (12
micrómetros convertido a cm)
    focal_length_width = mtx[0][0] * tamano_pixel_cm
    focal_length_height = mtx[1][1] * tamano_pixel_cm

    print("\nFocal width (cm):")
    print(focal_length_width)
    print("\nFocal height (cm):")
    print(focal_length_height)

    # Mostrar imagen con intersecciones detectadas
    cv2.imshow('Intersections', img_with_intersections)
```

```
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return mtx, dist

# Función principal para rastrear la trayectoria de la cámara
def track_camera_trajectory(video_path, calib_mtx, calib_dist):
    # Inicialización de la captura de video
    video_cap = cv2.VideoCapture(video_path)
    ret, last_frame = video_cap.read()

    # Inicialización de variables para la trayectoria de la
cámara
    t = np.zeros((3, 1))  # Vector de traslación acumulada
    camera_translation = []  # Lista para guardar las
traslaciones
    frame_count = 0

    # Inicialización de ORB y BFMatcher
    orb = cv2.ORB_create()
    matcher = cv2.BFMatcher()

    # Configuración de la visualización de la trayectoria en
vivo
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    plt.ion()  # Modo interactivo

    # Bucle principal para procesar cada cuadro del video
    while video_cap.isOpened():
        ret, frame = video_cap.read()
        if not ret:
```

```
            break

        frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        query_img = frame_bw.copy()
        train_img = last_frame.copy()

        # Detectar y describir características en el cuadro
actual y anterior
        queryKeypoints, queryDescriptors =
orb.detectAndCompute(query_img, None)
        trainKeypoints, trainDescriptors =
orb.detectAndCompute(train_img, None)

        # Emparejar las características entre cuadros
        matches = matcher.match(queryDescriptors,
trainDescriptors)
        matches = sorted(matches, key=lambda x:
x.distance)[:100] #100

        # Convertir los puntos emparejados en arrays numpy
        query_idx = [m.queryIdx for m in matches]
        train_idx = [m.trainIdx for m in matches]
        query_points = cv2.KeyPoint.convert(queryKeypoints,
query_idx)
        train_points = cv2.KeyPoint.convert(trainKeypoints,
train_idx)

        # Calcular la matriz esencial y recuperar la pose de la
cámara
        E, mask = cv2.findEssentialMat(query_points,
train_points, calib_mtx)
```

```
        _, R, T, mask = cv2.recoverPose(E, query_points,
train_points, calib_mtx)

        # Acumular la traslación para rastrear la trayectoria de
la cámara
        t += T
        camera_translation.append(t.copy())

        frame_count += 1
        if frame_count % 10 == 0:
            # Actualización de la gráfica en vivo
            pos = np.array(camera_translation)
            ax.clear()
            ax.plot(pos[:, 0, 0], pos[:, 1, 0], pos[:, 2, 0],
label='Camera Trajectory')
            ax.set_xlabel('X')
            ax.set_ylabel('Y')
            ax.set_zlabel('Z')
            plt.draw()
            plt.pause(0.01)

        # Visualización de las coincidencias
        img_matches = cv2.drawMatches(query_img, queryKeypoints,
train_img, trainKeypoints, matches, None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
        cv2.imshow('ORB Matches', img_matches)

        if cv2.waitKey(1) == ord(' '):
            break

        last_frame = frame_bw
```

```
    # Liberar recursos
    video_cap.release()
    cv2.destroyAllWindows()
    plt.ioff()
    plt.show()

if __name__ == "__main__":
    image_path = r'C:\Users\rodri\MR5
TrackingPatos\PatronLoza\Captura de pantalla 2024-05-15
222749.png'
    video_path = r'C:\Users\rodri\MR5
TrackingPatos\DuckVideo.mp4'

    # Calibrar la cámara desde una imagen
    calib_mtx, calib_dist =
calibrate_camera_from_image(image_path)

    # Rastrear la trayectoria de la cámara usando el video
    track_camera_trajectory(video_path, calib_mtx, calib_dist)
```

## Resultados

1. **Calibración de la Cámara:**
    - La calibración de la cámara fue exitosa, proporcionando una matriz de la cámara y coeficientes de distorsión precisos. La longitud focal calculada en centímetros fue utilizada para estimar distancias precisas a los patos detectados.
2. **Detección y Seguimiento de Patos:**
    - La detección de patos utilizando YOLOv5 fue efectiva, con el modelo identificando correctamente los patos en múltiples cuadros del video. La distancia a los patos detectados se calculó con precisión, mostrando los resultados en tiempo real.
3. **Generación de Modelos 2D y 3D:**
    - La imagen del pato detectado con mayor confianza fue extraída y guardada. A partir de esta imagen, se generó un modelo 3D utilizando la biblioteca Open3D. Los modelos 3D fueron guardados en formato .ply y pudieron ser visualizados interactivamente.
4. **Rastreo de la Trayectoria de la Cámara:**
    - Utilizando características ORB y coincidencias BFMatcher, se rastreó la trayectoria de la cámara a lo largo del tiempo. La trayectoria se mostró en una gráfica 3D interactiva, proporcionando una representación visual de los movimientos de la cámara durante la grabación del video.

## Limitaciones

1. **Precisión de la Detección:**
    - La precisión de la detección de patos depende en gran medida de la calidad del modelo YOLOv5 y la resolución del video. En condiciones de baja iluminación o con patos parcialmente ocultos, el modelo puede tener dificultades para detectar los objetos correctamente.
2. **Calibración de la Cámara:**
    - La precisión de la calibración puede verse afectada si los puntos de referencia en la imagen no son claramente identificables o si la imagen de referencia no es suficientemente clara.
3. **Generación de Modelos 3D:**
    - La calidad del modelo depende de la precisión de la imagen y de la detección de bordes. La densidad y precisión de la nube de puntos generada pueden verse limitadas por la resolución de la imagen y la precisión de las técnicas de procesamiento de imágenes utilizadas.
4. **Rastreo de la Trayectoria de la Cámara:**
    - La precisión del rastreo de la trayectoria de la cámara depende de la calidad de las características detectadas y emparejadas en los cuadros del video.
    - La visualización en vivo de la trayectoria puede ser limitada por la capacidad de procesamiento del sistema y la resolución del video, lo que puede afectar la fluidez de la representación gráfica.