Informe de Práctica 4: Seguimiento de Rostro con Dron Tello

Elaborado por: Daniel Francisco Calderón Lebro

1. Introducción

En esta práctica se implementó un Dron Tello para realizar un seguimiento de rostros mediante un programa en Python. El objetivo principal es utilizar algoritmos de visión por computadora para detectar y seguir un rostro en tiempo real, asegurando una distancia adecuada.

2. Descripción del Código

El código adjunto muestra la secuencia paso a paso para guiar el dron mediante diversos comandos. En primer lugar, se realiza la detección del rostro utilizando un algoritmo Haar Classifier previamente entrenado para la clasificación y detección de rostros en las imágenes capturadas por la cámara del dron. Luego, se procesa la imagen con OpenCV para calcular el centro de masa de esta y determinar la proximidad del rostro. Este valor de distancia se controla mediante un controlador PID.

3. Secuencia de Movimientos

El programa está diseñado para despegar el dron. Una vez que el dron está en el aire, se eleva a una distancia predeterminada, alcanzando una altura óptima para capturar el rostro de la persona de manera efectiva. A partir de ahí, comienza a transmitir imágenes al programa en Python, capturadas por la cámara del dron.

Cuando se detecta un rostro, el programa calcula el tamaño del cuadro generado por la detección del rostro. Utilizando constantes definidas, compara el tamaño de este cuadro con el tamaño de la imagen capturada por la cámara y establece un error que debe ser corregido para mantener una distancia moderada del rostro. Si el dron está demasiado cerca o lejos, ajusta su posición en consecuencia.

Además, el procesamiento de la imagen y la respuesta del dron se muestran en una ventana generada por el código en Visual Studio Code, lo que permite una visualización fluida del seguimiento del rostro en tiempo real. La imagen generada presenta un ejemplo del rostro capturado, marcado con un punto de centro de masa respectivo al tamaño del rostro.

Ejemplo real del funcionamiento del código

4. Código Implementado

python

```
import cv2
import numpy as np
from djitellopy import tello
import time


me = tello.Tello()
me.connect()
print(me.get_battery())
me.streamon()
me.takeoff()
me.send_rc_control(0, 0, 25, 0)
```

```
time.sleep(2.2)


w, h = 360, 240
fbRange = [6200, 6800]
pid = [0.4, 0.4, 0.001]
pError = 0


def findFace(img):
    faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGray, 1.2, 8)
    myFaceListC = []
    myFaceListArea = []
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cx = x + w // 2
        cy = y + h // 2
        area = w * h
        cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)
        myFaceListC.append([cx, cy])
        myFaceListArea.append(area)
    if len(myFaceListArea) != 0:
        i = myFaceListArea.index(max(myFaceListArea))
        return img, [myFaceListC[i], myFaceListArea[i]]
    else:
        return img, [[0, 0], 0]


def trackFace(info, w, pid, pError):
    area = info[1]
```

```
    x, y = info[0]
    fb = 0
    error = x - w // 2
    speed = pid[0] * error + pid[1] * (error - pError)
    speed = int(np.clip(speed, -100, 100))
    if area > fbRange[0] and area < fbRange[1]:
        fb = 0
    elif area > fbRange[1]:
        fb = -20
    elif area < fbRange[0] and area != 0:
        fb = 20
    if x == 0:
        speed = 0
        error = 0
    print(speed, fb)
    me.send_rc_control(0, fb, 0, speed)
    return error

# cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
while True:
    # _, img = cap.read()
    img = me.get_frame_read().frame
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (w, h))
    img, info = findFace(img)
    pError = trackFace(info, w, pid, pError)
    print("Center", info[0], "Area", info[1]) #
    cv2.imshow("Output", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
```

me.land()

break

5. Conclusión

Este programa permite al dron navegar de forma precisa y realizar un seguimiento óptimo del rostro capturado y presentado en pantalla. Se logra controlar una distancia adecuada del rostro para evitar incidentes. Los clasificadores Haar son algoritmos prediseñados para realizar tareas de clasificación y detección mediante machine learning con un bajo costo computacional.