**GUÍA VUELO INICIAL DE DRON TELLO POR PYTHON**

**INTRODUCCIÓN**

Esta guía está centrada en el estudio de las posibilidades ofrecidas por el drone Tello con un propósito experimental y educacional. El tello es una pequeña aeronave programable de DJI con unas características técnicas limitadas en cuanto a precisión y duración de batería, pero presenta las condiciones perfectas para poder explorar las diferentes capacidades de un drone mediante la programación. Para empezar, se va a explicar la manera de poder realizar movimientos básicos desde el ordenador. Entre ellas se encuentra como pilotar el drone mediante teclado, conseguir visualizar la imagen captada por la aeronave en directo o tomar fotografías. También se desarrollarán aplicaciones más complejas como el mapeo del movimiento del drone, el reconocimiento facial y el seguimiento de este.

**DRONE TELLO EDU**

Tello EDU es un pequeño cuadricóptero fabricado por Shenzhen Ryze Technology, con procesadores de Intel y control de vuelo incorporado por DJI. Esta aeronave es idónea para poder desarrollar infinidad de aplicaciones de manera educativa gracias a la facilidad que ofrece para ser programado mediante diferentes lenguajes de programación (Scratch, Python y Swift…)



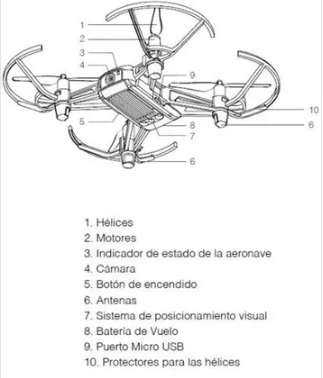
**COMPONENTES DEL DRONE**

En la siguiente imagen, podrá observar los diferentes componentes que posee el

Tello EDU. Es importante destacar que el sistema de posicionamiento visual hace

referencia a una cámara, de características técnicas inferiores a la frontal, que

puede ser ciertamente útil en algunos casos.



**INDICADOR DE ESTADO DEL DRONE**

El indicador de estado del Tello consiste en un LED ubicado en la parte

frontal de la aeronave, junto a la cámara delantera. Éste nos ofrece información

acerca de la condición del drone y su batería dependiendo del color y patrón de

repetición. En la siguiente tabla se definen los diferentes indicadores de estado.

|  | **COLOR** | **PATRÓN** | **ESTADO DEL DRONE** |
| --- | --- | --- | --- |
| Estados  normales | Alterna entre rojo, verde y amarillo | Parpadea | Encendido y realizando pruebas de autodiagnóstico |
| Verde | Parpadea dos veces de forma periódica | Sistema de posicionamiento visual activado |
| Amarillo | Parpadea lentamente | Sistema de posicionamiento visual no disponible, la aeronave se encuentra en modo Atti (Attitude mode) |
| Estados de advertencia | Amarillo | Parpadea rápidamente | Pérdida de señal del control remoto |
| Rojo | Parpadea lentamente | Advertencia de nivel |
| Rojo | Parpadea rápidamente | Batería baja crítica |
| Rojo | Luz fija | Error crítico |

**ESPECIFICACIONES TÉCNICAS**

En cuanto a las especificaciones técnicas, en la siguiente tabla se pueden observar las referentes al drone, la cámara o batería.

| Aeronave | Modelo | ------- |
| --- | --- | --- |
| Peso | 87 g |
| Dimensiones | 98x92,5x41 mm |
| Velocidad máxima | 28,8 km/h (8 m/s) |
| Tiempo de vuelo máx. | 13 minutos |
| Distancia de vuelo máx. | 100 m |
| Altura de vuelo máx. | 30 m |
| Intervalo de temperatura de funcionamiento | De 0°C a 40°C |
| Intervalo de frecuencias de funcionamiento | De 2,4 a 2,4835 GHz |
| Transmisor (PIRE) | 20 dBm (FCC) |
| 19 dBm (CE) |
| 19 dBm (SRRC) |
| Cámara | Tamaño imagen máximo | 2592x1936 |
| Formato de imagen | JPG |
| Modos de grabación de vídeo | HD: 1280x720 30p |
| Formato de vídeo | MP4 |
| Batería | Capacidad | 1100 mAh |
| Voltaje | 3,8 V |
| Tipo de batería | LiPo |
| Energía | 4,18 Wh |
| Potencia de carga máx. | 10 W |

**SDK 2.0**

El Tello Edu cuenta con un Kit de Desarrollo para el Software que sirve de base

para la creación de diferentes aplicaciones y funcionalidades. Al igual que el

manual de instrucciones comentado anteriormente, también existe un manual

de uso del SDK 2.0 donde se explica el funcionamiento de la aeronave

y cómo controlarlo mediante todo tipo de comandos.

Durante todo el trabajo se va a hacer uso de estos comandos ofrecidos por el

SDK, por lo que más adelante se entrará en detalle en ellos.

**App oficial Tello**

El Tello EDU puede ser controlado mediante la aplicación llamada Tello. Esta

app está desarrollada por Shenzhen RYZE Tech Co. Ltd. y gracias a ella se

puede pilotar fácilmente la aeronave, así como sacar partido a todas las

funcionalidades que ésta ofrece.

La interfaz principal ofrece los dos joysticks necesarios para pilotar el

drone, botones para iniciar el despegue, elegir entre los diferentes modos de

vuelo, acceder a la configuración, ver los archivos multimedia, seleccionar

fotografía o vídeo o para tomar fotos o iniciar la grabación, y finalmente,

también se puede observar cierta información acerca de la batería, la señal, la

posición y velocidad de la aeronave.



**MODOS DE VUELO**

La aplicación oficial de Tello ofrece diferentes modos de vuelo con

distintas funcionalidades como se puede observar en la siguiente imagen.

Posteriormente, se va a dar uso a algunas de estas capacidades que ofrece el

drone en las aplicaciones creadas, y, además, se desarrollarán nuevos modos

de vuelo como ya se ha comentado en la introducción.



Para empezar, cabe destacar que el trabajo ha sido desarrollado en Ubuntu 20.04, ya que es el sistema en el que mejor me desenvuelvo.

Por otro lado, se ha usado el lenguaje de programación Python, el cual ofrece un amplio abanico de posibilidades para sacar provecho al Tello.

**INSTALACIÓN DEL ENTORNO DE TRABAJO**

Como he comentado anteriormente, el lenguaje de programación usado

durante el trabajo es Python, por lo que es primordial la correcta instalación de

éste. Para esto, vamos a dirigirnos a la **TERMINAL** de Ubuntu y se agrega el

siguiente comando para la instalación de PyCharm Community.

sudo snap install pycharm-community –classic

Colocamos la contraseña invisible de nuestro equipo, en este caso, la contraseña de los equipos del laboratorio es 0000 (Cuatro ceros).

Finalmente, otro requisito importante es la instalación de la librería djitellopy,

donde todas las comandas que hacíamos mención en SDK 2.0 están incorporadas.

Mediante su Github, se puede observar la información necesaria acerca de esta

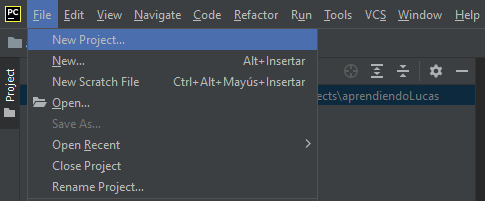
librería y las posibilidades que nos ofrece. Más adelante, dependiendo de las diferentes funcionalidades que se desarrollen, se tendrán que tener en cuenta otras librerías.

**CREAR UN NUEVO PROYECTO EN PYCHARM**

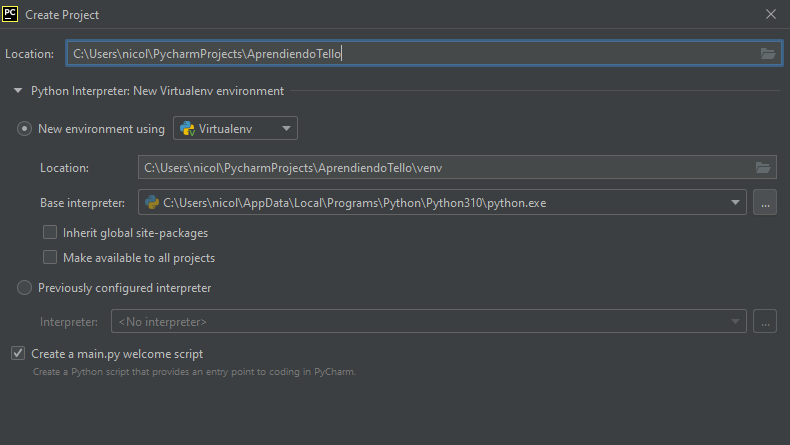
1. Iniciar el programa Pycharm



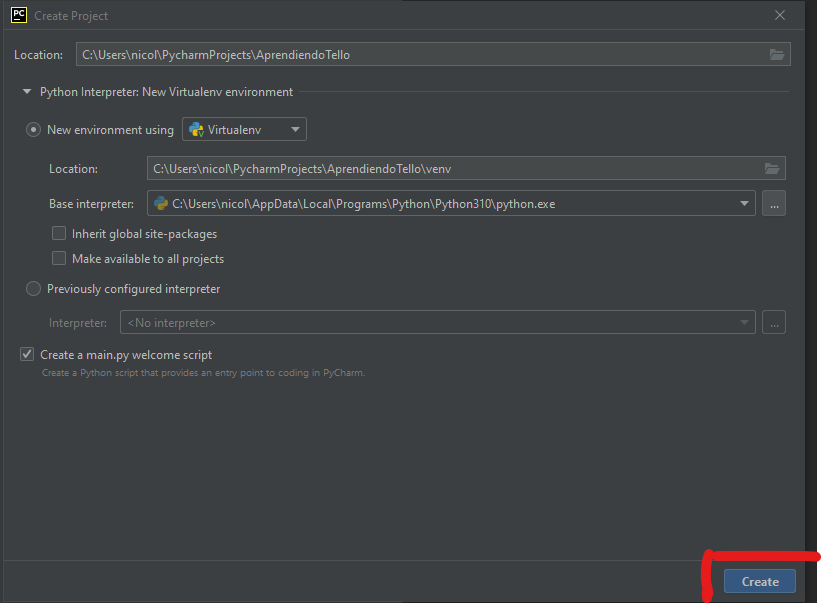
2. En menú File, Click en la opción New Proyect.



3. Colocamos un nombre a nuestro Proyecto, en este caso, lo llamaré “AprendiendoTello”.

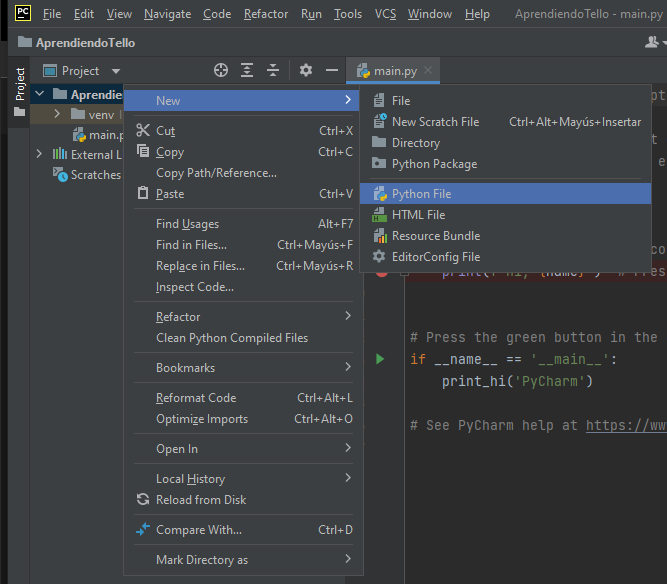


4. Una vez nombrado nuestro proyecto, procedemos a dar click en el botón Create.

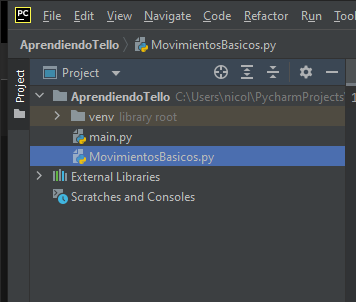


Una vez creado nuestro proyecto, vamos a crear un archivo donde almacenaremos todo nuestro código. Para crear un archivo en nuestro proyecto, siga estos pasos:

1. Damos click derecho sobre nuestro proyecto “AprendiendoTello”, nos aparecerá una opción llamada “New”, desplegamos su lista y nos aparecerá una opción llamada Python File, damos click.



2. Damos un nombre a nuestro archivo, en este caso lo llamaré “Movimientos básicos” y damos Enter. Nos debe quedar así:



Una vez hecho esto, ya estamos listos para hacer nuestro código de movimientos básicos de nuestro Tello drone.

**MOVIMIENTOS BÁSICOS**

Para empezar, es importante conocer la manera de conectar el drone al

ordenador. Para ello, simplemente hay que encenderlo, conectarnos desde el

PC a su conexión Wi-Fi (Los equipos del laboratorio requieren de una antena, por

favor solicite al profesor o asistente la respectiva antena) y realizar la comanda

connect(). En mi caso, siempre pido que me indique la batería del drone mediante

la comanda get\_battery(), ya que es una información vital a la hora de trabajar

con él, y, además, nos aseguramos de que la conexión se ha realizado con éxito al

recibir respuesta.

Seguidamente, se debe conocer que para despegar se usa la comanda

takeoff() y para aterrizar land().

Por último, falta por indicar las comandas de movimiento básicas. Por un lado,

podemos indicar la dirección de movimiento o de giro y los centímetros o

grados deseados utilizando comandas como move\_up(100), move\_rigth(30),

move\_fordward(40), rotate\_clockwise(50) y sus comandas opuestas.

Por otro lado, se puede usar la comanda send\_rc\_control(), donde es necesario

indicar las velocidades de todas las componentes de movimiento. De esta

manera, podemos indicar, por ejemplo, que se mueva a 50 cm/s hacia delante

(send\_rc\_control(0, 50, 0, 0)). Si por el contrario queremos que se mueva hacia

atrás, deberíamos enviar la comanda (send\_rc\_control(0, -50, 0, 0).

**Código movimientos básicos:**

from djitellopy import tello

from time import sleep

me = tello.Tello()

me.connect()

print(me.get\_battery())

me.takeoff()

#Go forward/backward

me.send\_rc\_control(0, 50, 0, 0)

sleep(2) #Tiempo de espera entre cada comando

#Go right/left

me.send\_rc\_control(30, 0, 0, 0)

sleep(2)

#Rotate clockwise/anticlockwise (yaw)

me.send\_rc\_control(0, 0, 0, 30)

sleep(2)

#Go up/down

me.send\_rc\_control(0, 0, 30, 0)

sleep(2)

me.send\_rc\_control(0, 0 , 0, 0)

me.land()

**TRANSMISIÓN DE VÍDEO**

Para poder obtener en nuestro ordenador la imagen captada por el drone, se

va a hacer uso de la librería opencv-python, que es una de las librerías que

mejor funciona para el procesado de imagen.

Una vez conectados al drone como se ha indicado anteriormente, el comando

necesario para empezar a transmitir es streamon(), así como streamoff() para

finalizar. Además, se va a hacer uso de la comanda get\_frame\_read().frame,

que nos ofrece el frame recibido por el drone actualizado en todo momento, es

decir, la imagen que queremos observar. Cabe destacar que hay que añadir un

pequeño delay de milisegundos (cv2.waitKey(1)), de no ser así, no podríamos

ver la imagen.

Por último, se hará uso de opencv para tratar esta imagen. En mi caso,

recomiendo usar la función cv2.resize(), ya que de no ser así la ventana donde

podemos observar la imagen mediante cv2.imshow(), es muy grande y puede

que nos afecte en el procesado de ésta dependiendo de la capacidad del

ordenador.

**Código captura de imagen:**

from djitellopy import tello

import cv2

me = tello.Tello()

me.connect()

print(me.get\_battery())

me.streamon()

while True:

img = me.get\_frame\_read().frame

img = cv2.resize(img, (360, 240))

cv2.imshow(“Image”, img)

cv2.waitKey(1)

**CONTROL DESDE TECLADO Y TOMA DE IMÁGENES**

El control del drone así como la posibilidad de poder tomar fotos son

funciones esenciales para cualquier aplicación de control de drones. Para

poder desarrollar estas funciones, se va a hacer uso de la librería pygame, así

como las mencionadas anteriormente. Primeramente, el objetivo es crear un

módulo donde incluir todo el procesado del teclado para poder utilizarlo

siempre que se desee más adelante con facilidad.

A la hora de crear el módulo, hay que tener en cuenta que pygame se basa en

el desarrollo de juegos, por lo que es necesario tener una ventana de juego

siempre disponible para que se detecten las teclas presionadas del teclado.

Mediante la función pygame.display.set\_mode() e indicando el tamaño deseado

para la ventana, se crea.

Posteriormente, es necesario crear una función que indique si existe una tecla

presionada. Para comprobar si esto está sucediendo, se hace uso de

pygame.key.get\_pressed(). Es importante saber que se debe seguir un formato

para cada tecla presionada en esta librería. Por ejemplo, si la tecla “UP” está

presionada, se debe tratar como K\_UP

**Módulo procesado de teclado:**

import pygame

def init():

pygame.init()

#Windows Game creation

win = pygame.display.set\_mode((400, 400))

def getKey(keyName):

ans = False

for eve in pygame.event.get(): pass

keyInput = pygame.key.get\_pressed()

myKey = getattr(pygame, ‘K\_{}’.format(keyName))

if keyInput[myKey]:

ans = True

pygame.display.update()

return ans

Una vez creado el módulo, ya se puede usar para generar una pequeña aplicación

que permita controlar el drone utilizando el teclado y tomar fotos.

Inicialmente, se deben importar todas las librerías necesarias, además del

nuevo módulo, y realizar la conexión a la aeronave junto con el inicio de

retransmisión de video.

A continuación, se va a crear una función donde se definirá la velocidad de

movimiento que se desee (50 cm/s en este caso) y dependiendo de la tecla

presionada, se devolverán los parámetros de velocidad en dirección izquierda o

derecha (lr), delante o detrás (fb), arriba o abajo (ud) y giro en sentido horario o

antihorario (yv). Por ejemplo, si se presiona la tecla “LEFT”, el parámetro lr

deberá ser -50 y todos los demás 0.

Una vez obtenidos estos parámetros, sólo hay que aplicar la comanda que se

utilizó en Movimientos Básicos de la siguiente manera:

send\_rc\_control(lr, fb, ud, yv)

Hay que destacar que este modo de aplicar el movimiento es interesante y

eficiente, ya que se pueden presionar varias teclas al mismo tiempo sin que

suponga ningún problema, porque en el comando send\_rc\_control(), se tienen

que definir las velocidades en las 4 direcciones siempre.

Por otro lado, hay que incorporar que al pulsar la tecla “q” se realice la

comanda land() y mediante la tecla “e” takeoff().

Para terminar, presionando la tecla “z”, se va a realizar la función de

opencv-python imwrite(f'Resources/Images/{time.time()}.jpg', img). Es decir, se

van a tomar imágenes de la retransmisión en vídeo y se guardarán en el

directorio Resources/Images (creado anteriormente) en formato jpg utilizando

como nombre de archivo el tiempo en que se tomó la foto. De esta manera,

nunca se van a sobrescribir las imágenes al ser tomadas en diferentes

instantes y tener nombres diferentes.

Es importante dejar un pequeño delay después de presionar la tecla “z” para

tomar fotos, ya que, de no ser así, se van a guardar una gran cantidad de

fotografías iguales.

**Control solo teclado:**

Es importante resaltar que debes nombrar este archivo como “teclado” ya que lo usaremos para el código de teclado con cámara.

import pygame

def init ():

pygame.init()

win = pygame.display.set\_mode((400,400))

def getKey(keyName):

ans = False

for eve in pygame.event.get(): pass

keyInput = pygame.key.get\_pressed()

myKey = getattr(pygame, 'K\_{}'.format(keyName))

if keyInput[myKey]:

ans = True

pygame.display.update()

return ans

def main():

if getKey("LEFT"):

print("Left key pressed")

if getKey("RIGHT"):

print("Right key pressed")

if \_\_name\_\_=='\_\_main\_\_':

init()

while True:

main()

**Control desde teclado y toma de fotos:**

from djitellopy import tello

import teclado as kp

from time import sleep, time

import cv2

kp.init()

me = tello.Tello()

me.connect()

print(me.get\_battery())

global img

me.streamon()

def getKeyboardInput():

lr, fb, ud, yv = 0, 0, 0, 0

speed = 50

if kp.getKey("LEFT"): lr= -speed

elif kp.getKey("RIGHT"): lr = speed

if kp.getKey("UP"): fb = -speed

elif kp.getKey("DOWN"): fb = speed

if kp.getKey("w"): ud= speed

elif kp.getKey("s"): ud = -speed

if kp.getKey("a"): yv= speed

elif kp.getKey("d"): yv = -speed

if kp.getKey("q"): yv = me.land(); time.sleep(3)

if kp.getKey("e"): yv = me.takeoff()

if kp.getKey("z"):

cv2.imwrite(f'Resources/Images/{time.time()}.jpg',img)

time.sleep(0.3)

return [lr, fb, ud, yv]

me.takeoff()

while True:

vals = getKeyboardInput()

me.send\_rc\_control(vals[0], vals[1], vals[2], vals[3])

img = me.get\_frame\_read().frame

img = cv2.resize(img, (360, 240))

cv2.imshow("Image", img)

cv2.waitKey(1)

sleep(0.05)

Desde mi punto de vista, el desarrollo de esta funcionalidad no ha sido muy

complejo, me ha ayudado a entender la eficacia que nos aportan los módulos

creados y la manera en que proceder para dar comandos al drone

dependiendo de las teclas presionadas. En contraparte, la necesidad de tener

una ventana de juego de pygame únicamente para que detecte las teclas

presionadas me ha parecido poco eficiente.

**MAPEO DEL MOVIMIENTO**

La siguiente aplicación que queremos desarrollar es el mapeo de todo el

movimiento realizado por nuestro drone en 2D.

También, se va a realizar un estudio propio acerca de la precisión en la

velocidad lineal y angular del Tello EDU y se corregirá un error cometido en el

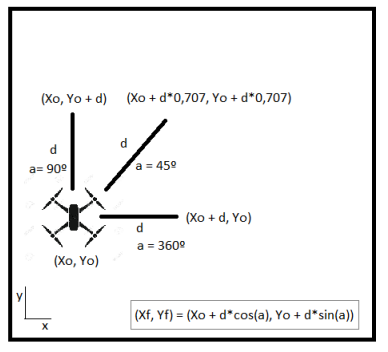
tutorial. Esta equivocación en el código propuesto mapeaba erróneamente el

movimiento del drone al no haber tenido en cuenta los movimientos diagonales

que involucran más de una tecla pulsada al mismo tiempo.

Para empezar, se va realizar una pequeña explicación teórica de cómo vamos

a afrontar la codificación de esta funcionalidad.



Como se puede observar en la imagen, nuestro drone siempre va a partir

de una posición inicial (Xo, Yo), específicamente de donde se ha despegado.

Teniendo en cuenta que se puede fijar la velocidad lineal y angular de nuestro

drone y que se programa cada cuanto tiempo se va a chequear la posición,

realizar el producto de ambas constantes (v\*t) nos ofrecerá la distancia

recorrida (“d”) o los grados girados en el intervalo de tiempo programado.

Al ya conocer estas constantes, mediante trigonometría y siempre teniendo en

cuenta los valores de “x” e “y” anteriores, se va a poder obtener la posición del

cuadricóptero en todo momento.

Una vez realizado el estudio teórico de la metodología a seguir, se va a explicar

su posterior codificación.

Para poder desarrollar el código, es necesario importar las librerías numpy y

math a las mencionadas anteriormente. También será necesario el uso del

módulo y prácticamente la totalidad del código creado, el Control desde

teclado y toma de imágenes, ya que se busca seguir moviendo el drone usando

el teclado y que este movimiento quede mapeado.

A continuación, va a ser necesario fijar la velocidad lineal y angular, así como el

intervalo de tiempo deseado. Teóricamente, hasta ahora se había visto que si

se usa la comanda, por ejemplo, send\_rc\_control(0, 0, -50, 0), el drone

desciende a 50 cm/s. En la práctica, esta velocidad nunca es constante, ya que

el Tello no es muy preciso, por lo que si hacemos los cálculos de posición

con una velocidad teórica, en la práctica el drone no estará ubicado donde

indica el mapa.

Por ello, se va a realizar un pequeño estudio buscando el valor más real

posible de velocidad. Se fijará la velocidad teórica del drone a 15 cm/s y se le

ordenará que avance a esa velocidad durante 10 segundos. Posteriormente, se

medirá la distancia real que ha recorrido en ese tiempo obteniendo una

velocidad real (distancia/tiempo). Este proceso se repetirá 10 veces, por la

poca precisión del Tello EDU y finalmente se promediarán todos los resultados.

**Estudio de velocidad lineal real:**

| Distancia recorrida en 10s [cm] | Velocidad Real [cm/s] |
| --- | --- |
| 160 | 16 |
| 135 | 13,5 |
| 120 | 12 |
| 137 | 13,7 |
| 119 | 11,9 |
| 206 | 20,6 |
| 245 | 24,5 |
| 138 | 13,8 |
| 144 | 14,4 |
| 245 | 24,5 |

Una vez realizado el estudio, se ha obtenido un valor promediado de 16,5 cm/s.

A primera instancia, parece un resultado real relativamente parecido a los 15

cm/s teóricos, pero si calculamos su desviación típica se obtiene un valor de

11,42 cm, es decir, de media el resultado será 11,42 centímetros erróneo.

Además, hay casos en que la velocidad real es casi 10 cm/s superior a la

teórica. En el tutorial seguido, el valor promedio es de 11,7 cm/s, por lo que

también se puede apreciar la poca precisión del drone comparando los

resultados finales de ambos estudios.

Para la velocidad angular, se va a realizar el mismo procedimiento fijando la

velocidad angular teórica a 50 º/s, pero en este caso, se medirá el tiempo que

tarda el drone en dar una vuelta completa. De esta manera, no se tendrá que

medir el ángulo de giro realizado, que es una medición aproximada y poco

precisa. El valor promedio obtenido fue 9,98 segundos, con una desviación

típica muy pequeña, es decir, todos los resultados son iguales. Este valor

implica una velocidad angular real de 36 º/s aproximadamente, bastante

diferente a los 50 º/s teóricos. En el caso del tutorial, los resultados obtenidos

son iguales a los realizados en este caso.

**Estudio de la velocidad angular real:**

| Tiempo en completar 360° [s] | Velocidad Real [grados/segundos] |
| --- | --- |
| 9,95 | 36,18 |
| 10 | 36 |
| 9.98 | 36,07 |
| 9,83 | 36,62 |
| 10.03 | 35,89 |
| 9,91 | 36,32 |
| 9,94 | 36,21 |
| 10,01 | 35,95 |
| 10 | 36 |
| 9,97 | 36,1 |

El siguiente paso es indicar los valores de “a” para cada tecla. Por

ejemplo, al pulsar la tecla “RIGHT”, indicaremos que “a” es 360º ya que

no tenemos componente vertical (sin(360º) = 0), por otro lado, si por

ejemplo pulsamos la tecla “DOWN”, el drone deberá retroceder, por lo

que “a” tiene que ser igual a 270º, siendo la componente “y” negativa y

obteniendo lógicamente un valor inferior en el sumatorio de la posición

vertical.

Finalmente, mediante la librería opencv-python, crearemos pequeños círculos rojos para cada punto recorrido y un círculo verde para la posición actual.

Además, incorporaremos los valores de “x” e “y” en la ubicación en que nos encontramos. Es interesante saber que en opencv las representaciones de color son en BGR, por lo que el color rojo será (0, 0, 255).

**Código completo:**

from djitellopy import tello

import teclado as kp

import numpy as np

import cv2

import math

from time import sleep

############PARÁMETROS###############

fSpeed = 165/10 #Velocidad lineal real

aSpeed = 36 #Velocidad angular real

Interval = 0.25 #Intervalo de tiempo

dInterval = fSpeed \* Interval #Distancia recorrida durante un intervalo

aInterval = aSpeed \* Interval #Grados girados durante un intervalo

x, y = 500, 500 #x0 y0

a = 0

yaw = 0

kp.init()

me.tello.Tello()

me.connect()

print(me.get\_battery())

points = [(0, 0), (0, 0)]

def getKeyboardInput():

lr, fb, ud, yv = 0, 0, 0, 0

aspeed = 50 #Velocidad Angular teórica

speed = 15 #Velocidad lineal teórica

global x, y, yaw, a

d = 0

if kp.getKey(“LEFT”):

lr = -speed

d = dInterval

a = 180

elif kp.getKey(“RIGHT”):

lr = speed

d = dInterval

a = 360

if kp.getKey(“UP”):

fb = speed

d = -dInterval

a = 90

elif kp.getKey(“DOWN”):

fb= -speed

d =-dInterval

a = 270

if kp.getKey(“w”):

ud = speed

elif kp.getKey(“s”):

ud = -speed

if kp.getKey(“a”):

yv = -aspeed

yaw = aInterval

elif kp.getKey(“d”):

yv = aspeed

yaw += aInterval

if kp.getKey(“q”): me.land()

if kp.getKey(”e”): me.takeoff()

sleep(interval)

#Sumatoria de las posiciones

a += yaw

x += int(d \* math.cos(math.radians(a)))

y += int(d \* math.sin(math.radians(a)))

return [lr, fb, ud, yv, x, y]

def drawPoints(img, points):

for point in points:

cv2.circle(img, point, 5, (0, 0, 255), cv2.FILLED) #Circulos rojos para cada punto

cv2.circle(img, points[-1], 8, (0, 255, 0), cv2.FILLED) #Circulos verdes para posición actual

cv2.putText(img, f’({(points[-1][0]-500)/100}, {(points[-1][-1]-500)/100})m’,

(points[-1][0]+ 10, points[-1][1]+30), cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 0, 255), 1)

while True:

vals = getKeyboardInput()

me.send\_rc\_control(vals[0], vals[1], vals[2], vals[3])

img = np.zeros((1000, 1000, 3), np.uint8) #Creación de imagen negra (mapa) como matriz de 0

#y valores irán de 0 a 256 (2^8) como integers sin signo

if (points [-1][0] != vals[4] or points [-1][1] != vals [5]): #Solo dibuja si se mueve

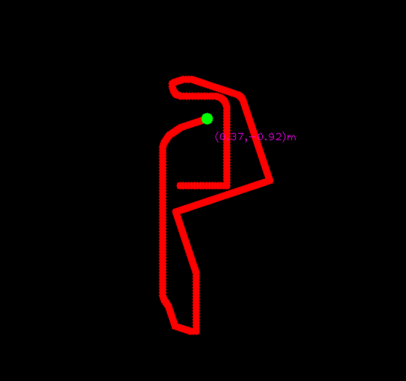
points.append((vals[4], vals[5]))

drawPoints(img, points)

cv2.imshow(“Output”, img)

cv2.waitKey(1)

**Interfaz mapeo:**

****

Para finalizar, me gustaría comentar que he encontrado un pequeño error en el vídeo Tutorial. A la hora de querer hacer volar el drone en diagonal, dos

teclas deben ser pulsadas, por ejemplo, “UP” y “LEFT”. Así que, con el modo

en que está desarrollado el código sólo se definiría un único valor

de “a”. Es decir, el drone volaría en diagonal, pero en el mapeo aparecería

volando hacia delante o hacia la izquierda. Para corregirlo, es necesario añadir

los ángulos para cada una de las posibles diagonales, asimismo, otros pequeños

cambios serán necesarios para el correcto funcionamiento de la aplicación en cuanto a los movimientos en diagonal.

**Definición de ángulos para diagonales:**

if kp.getKey(“LEFT”) and kp.getKey(“UP”):

a = 135

fb = speed

lr = -speed

d = dInterval

elif kp.getKey(“RIGHT”) and kp.getKey(“UP”):

a = 45

fb = speed

lr = speed

d = dInterval

elif kp.getKey(“LEFT”) and kp.getKey(“DOWN”):

a = 225

fb = -speed

lr = -speed

d = dInterval

elif kp.getKey(“RIGHT”) and kp.getKey(“DOWN”):

a = 315

fb = -speed

lr = speed

d = dInterval

Para dar por terminado este apartado, me gustaría comentar que esta función

me ha parecido muy interesante, ya que se puede extrapolar a poder chequear

todo el recorrido que ha realizado tu aeronave en un mapa. Además, esta

dinámica posibilita el conocimiento del punto de despegue, por lo que a pesar

de la poca precisión, se podría llegar a programar un modo Return to Home

(RTH) como en los drones más profesionales. Este modo es vital en el uso de

estas aeronaves, ya que, en caso de pérdida de cobertura, el drone volverá a su

punto de despegue.

**RECONOCIMIENTO FACIAL Y SEGUIMIENTO**

La siguiente funcionalidad que se ha desarrollado consiste en el

reconocimiento facial y posterior seguimiento de éste, es decir, se va a

programar que nuestro drone nos reconozca la cara y nos siga.

Primero, como ya se ha hecho en el apartado anterior, se va a explicar la teoría

que hay detrás del código, para que así sea más fácil la compresión de éste.

Mediante la librería opencv-python, se pueden reconocer caras y marcarlas con

un rectángulo, más adelante se entrará en detalle en cómo hacerlo. Como se

puede ver en la imagen, si la cara está más alejada, el área del

correspondiente rectángulo será más pequeño, y si esta área es inferior a un

valor establecido se le indicará al drone que avance. Si por el contrario el área

es superior al valor establecido, se le indicará que retroceda. En el caso que la

aeronave procese un área que se encuentre entre el límite de áreas inferior y

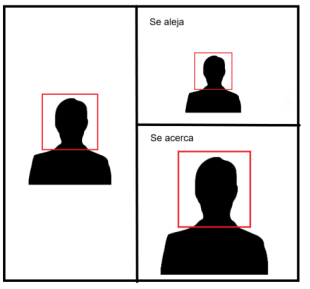
superior, no se moverá.

Esto puede desencadenar en un sistema inestable y no parar de realizar

movimientos hacia delante y hacia atrás al no conseguir que el área esté dentro

del rango deseado, por eso, se implementará un controlador PD que tendrá en

cuenta el error de área actual y anterior para evitar el balanceo del drone.



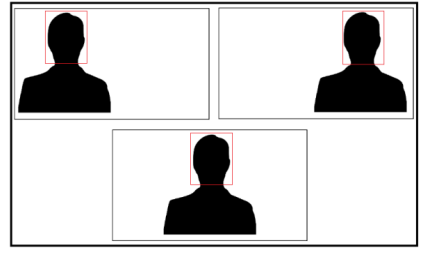
Por otro lado, como se puede observar en la otra imagen, si la cara no está

centrada en la imagen, se le ordenará al drone realizar movimientos de guiñada

hasta que ésta lo esté. En este caso, también implementaremos un controlador

PD para evitar un continuo movimiento de guiñada al no conseguir centrar la

cara.



Para el correcto funcionamiento del código, va a ser necesario importar las

librerías que se vienen utilizando hasta ahora, opencv-python, numpy y

djitellopy. Posteriormente, se definirá el ancho y alto de la imagen, que nos

servirá para conocer cuando la cara está centrada, el rango de áreas del

rectángulo, para que el drone sepa cuándo y cómo moverse dependiendo de la

proximidad a la cara, y los valores para los controladores PD, que han sido

fijados buscando el mejor rendimiento mediante prueba y error.

Para la detección facial, haremos uso del algoritmo de Viola-Jones, que tiene

una probabilidad de detección verdadera del 99,9% y tiene un coste

computacional muy bajo. Este método consiste en la división de la imagen en

subregiones de tamaños diferentes y gracias a una serie de clasificadores en

cascada, determina si esta subregión pertenece a una cara o no. Además,

mediante el uso de opencv, se puede implementar a nuestro código fácilmente.

Para ello, es necesario importar el código de los controladores en cascada, que

se puede encontrar en este drive:

<https://drive.google.com/file/d/1w4kqpVx1KGt4ziZuCgTv9S1mz8HBaNba/view?usp=sharing>

Este archivo se debe guardar en el directorio que se creó antes, el “Resources”.

Posteriormente, mediante opencv se marcarán las caras detectadas con

rectángulos rojos y su punto central con un punto verde. Pero el tello

únicamente se fijará en la cara con mayor área, es decir, la más cercana.

def findFace(img):

faceCascade=cv.CascadeClassifier(“Resources/haarcascade\_frontalface\_default.xml”)

imgGray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

faces = faceCascade.detectMultiScale(imgGray, 1.2, 8) #Detecta objetos y los retorna como una lista de rectángulos.

myFaceListC = [] #Creación de centros de caras

myFaceListArea = [] #Creación de lista de áreas

for (x, y, w, h) in faces:

cv2.rectangle(img, (x, y) (x+w, y+h), (0, 0, 255), 2) #Rectángulo rojo que marque la cara

cx = x + w // 2

cy = y + h // 2

area = w\*h

cv2.circle(img, (cx, cy), 5,(0, 255, 0), cv2.FILLED) #Circulo verde que marque el centro de la cara

myFaceListC.append([cx, cy])

myFaceListArea.append(area)

if len(myFaceListArea) != 0:

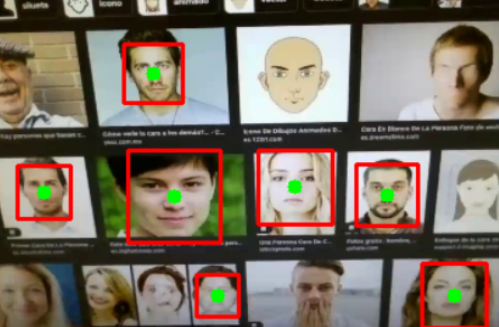
i = myFaceListArea.index(max(myFaceListArea)) #Únicamente queremos la cara más cercana

return img, [myFaceListC[i]]

else:

return img, [[0, 0], 0]

Al ejecutar este código, debe verse así:



Seguidamente, se programará la función para el seguimiento de la cara por

parte del drone. Para el correcto funcionamiento de ésta, es

necesario pasarle ciertos inputs: los referentes al drone para que se puedan

enviar comandos, la información de área y centro de ésta, la anchura de

nuestra imagen para saber si la cara está centrada, los parámetros de los

controladores PD y los valores de error anteriores.

def trackFace(me, info, w, pid, pid2, pError, pErrorFB)

area = info[1]

x, y = info[0]

fb = 0

ud = 0

#Cálculo de la yaw speed dependiendo del error de centrado actual y anterior

error = x – w // 2

speed = pid[0] \* error + pid[1]\*(error-pError)

speed = int(np.clip(speed, -50, 50)) #min. fijado a -50 y máx. a 50

#Cálculo de la fb speed dependiendo del error de area actual y anterior

if area < fbRange[0]:

errorFB = fbRange[0] - area

elif area > fbRange[1]:

errorFB = fbRange[1] – area

else:

errorFB = 0

speedFB = pid2[0] \* errorFB + pid2[1]\*(errorFB - pErrorFB)

speedFB = int(np.clip(speedFB, -50, 50))

if x == 0: #Si no hay cara detectada

speed = 0

error = 0

speed FB = 0

me.send\_rc\_control(0, speedFB, 0, speed)

return error, errorFB

Y bueno, se ha podido ver la programación que hay detrás de los modos de

vuelo de seguimiento de los drones profesionales. Estos modos se basan en

reconocer a personas y seguirlas, es decir, la misma dinámica que la aplicación

trabajada.