Help on Tello in module djitellopy.tello object:

class Tello(builtins.object)

| Tello(host='192.168.10.1', retry_count=3)

|

| Python wrapper to interact with the Ryze Tello drone using the official Tello api.

| Tello API documentation:

| [1.3](https://dl-cdn.ryzerobotics.com/downloads/tello/20180910/Tello%20SDK%20Documentation%20EN_1.3.pdf),

| [2.0 with EDU-only commands](https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf)

|

| Methods defined here:

|

| __del__(self)

|

| __init__(self, host='192.168.10.1', retry_count=3)

| Initialize self. See help(type(self)) for accurate signature.

|

| connect(self, wait_for_state=True)

| Enter SDK mode. Call this before any of the control functions.

|

| connect_to_wifi(self, ssid: str, password: str)

| Connects to the Wi-Fi with SSID and password.

| After this command the tello will reboot.

| Only works with Tello EDUs.

|

| curve_xyz_speed(self, x1: int, y1: int, z1: int, x2: int, y2: int, z2: int, speed: int)

| Fly to x2 y2 z2 in a curve via x2 y2 z2. Speed defines the traveling speed in cm/s.

|

| - Both points are relative to the current position

| - The current position and both points must form a circle arc.

| - If the arc radius is not within the range of 0.5-10 meters, it raises an Exception

| - x1/x2, y1/y2, z1/z2 can't both be between -20-20 at the same time, but can both be 0.

|

| Arguments:

| x1: -500-500

| x2: -500-500

| y1: -500-500

| y2: -500-500

| z1: -500-500

| z2: -500-500

| speed: 10-60

|

| curve_xyz_speed_mid(self, x1: int, y1: int, z1: int, x2: int, y2: int, z2: int, speed: int, mid: int)

| Fly to x2 y2 z2 in a curve via x2 y2 z2. Speed defines the traveling speed in cm/s.

|

| - Both points are relative to the mission pad with id mid.

| - The current position and both points must form a circle arc.

| - If the arc radius is not within the range of 0.5-10 meters, it raises an Exception

| - x1/x2, y1/y2, z1/z2 can't both be between -20-20 at the same time, but can both be 0.

|

| Arguments:

| x1: -500-500

| y1: -500-500

| z1: -500-500

| x2: -500-500

| y2: -500-500

| z2: -500-500

| speed: 10-60

| mid: 1-8

|

| disable_mission_pads(self)

| Disable mission pad detection

|

| emergency(self)

| Stop all motors immediately.

|

| enable_mission_pads(self)

| Enable mission pad detection

|

| end(self)

| Call this method when you want to end the tello object

```
|
| flip(self, direction: str)
|     Do a flip maneuver.
|     Users would normally call one of the flip_x functions instead.
|     Arguments:
|         direction: l (left), r (right), f (forward) or b (back)
|
| flip_back(self)
|     Flip backwards.
|
| flip_forward(self)
|     Flip forward.
|
| flip_left(self)
|     Flip to the left.
|
| flip_right(self)
|     Flip to the right.
|
| get_acceleration_x(self) -> float
|     X-Axis Acceleration
|     Returns:
|         float: acceleration
|
| get_acceleration_y(self) -> float
|     Y-Axis Acceleration
|     Returns:
|         float: acceleration
|
| get_acceleration_z(self) -> float
|     Z-Axis Acceleration
|     Returns:
|         float: acceleration
|
| get_barometer(self) -> int
|     Get current barometer measurement in cm
|     This resembles the absolute height.
|     See https://en.wikipedia.org/wiki/Altimeter
|     Returns:
|         int: barometer measurement in cm
```

|

| get_battery(self) -> int

| Get current battery percentage

| Returns:

| int: 0-100

|

| get_current_state(self) -> dict

| Call this function to attain the state of the Tello. Returns a dict

| with all fields.

| Internal method, you normally wouldn't call this yourself.

|

| get_distance_tof(self) -> int

| Get current distance value from TOF in cm

| Returns:

| int: TOF distance in cm

|

| get_flight_time(self) -> int

| Get the time the motors have been active in seconds

| Returns:

| int: flight time in s

|

| get_frame_read(self) -> 'BackgroundFrameRead'

| Get the BackgroundFrameRead object from the camera drone. Then, you just need to call

| backgroundFrameRead.frame to get the actual frame received by the drone.

| Returns:

| BackgroundFrameRead

|

| get_height(self) -> int

| Get current height in cm

| Returns:

| int: height in cm

|

| get_highest_temperature(self) -> int

| Get highest temperature

| Returns:

| float: highest temperature (°C)

|

| get_lowest_temperature(self) -> int

| Get lowest temperature

```
|     Returns:
|         int: lowest temperature (°C)
|
| get_mission_pad_distance_x(self) -> int
|     X distance to current mission pad
|     Only available on Tello EDUs after calling enable_mission_pads
|     Returns:
|         int: distance in cm
|
| get_mission_pad_distance_y(self) -> int
|     Y distance to current mission pad
|     Only available on Tello EDUs after calling enable_mission_pads
|     Returns:
|         int: distance in cm
|
| get_mission_pad_distance_z(self) -> int
|     Z distance to current mission pad
|     Only available on Tello EDUs after calling enable_mission_pads
|     Returns:
|         int: distance in cm
|
| get_mission_pad_id(self) -> int
|     Mission pad ID of the currently detected mission pad
|     Only available on Tello EDUs after calling enable_mission_pads
|     Returns:
|         int: -1 if none is detected, else 1-8
|
| get_own_udp_object(self)
|     Get own object from the global drones dict. This object is filled
|     with responses and state information by the receiver threads.
|     Internal method, you normally wouldn't call this yourself.
|
| get_pitch(self) -> int
|     Get pitch in degree
|     Returns:
|         int: pitch in degree
|
| get_roll(self) -> int
|     Get roll in degree
|     Returns:
```

```
|         int: roll in degree
|
| get_speed_x(self) -> int
|     X-Axis Speed
|     Returns:
|         int: speed
|
| get_speed_y(self) -> int
|     Y-Axis Speed
|     Returns:
|         int: speed
|
| get_speed_z(self) -> int
|     Z-Axis Speed
|     Returns:
|         int: speed
|
| get_state_field(self, key: str)
|     Get a specific sate field by name.
|     Internal method, you normally wouldn't call this yourself.
|
| get_temperature(self) -> float
|     Get average temperature
|     Returns:
|         float: average temperature (°C)
|
| get_udp_video_address(self) -> str
|     Internal method, you normally wouldn't call this youself.
|
| get_video_capture(self)
|     Get the VideoCapture object from the camera drone.
|     Users usually want to use get_frame_read instead.
|     Returns:
|         VideoCapture
|
| get_yaw(self) -> int
|     Get yaw in degree
|     Returns:
|         int: yaw in degree
|
```

| go_xyz_speed(self, x: int, y: int, z: int, speed: int)

| Fly to x y z relative to the current position.

| Speed defines the traveling speed in cm/s.

| Arguments:

| x: -500-500

| y: -500-500

| z: -500-500

| speed: 10-100

|

| go_xyz_speed_mid(self, x: int, y: int, z: int, speed: int, mid: int)

| Fly to x y z relative to the mission pad with id mid.

| Speed defines the traveling speed in cm/s.

| Arguments:

| x: -500-500

| y: -500-500

| z: -500-500

| speed: 10-100

| mid: 1-8

|

| go_xyz_speed_yaw_mid(self, x: int, y: int, z: int, speed: int, yaw: int, mid1: int, mid2: int)

| Fly to x y z relative to mid1.

| Then fly to 0 0 z over mid2 and rotate to yaw relative to mid2's rotation.

| Speed defines the traveling speed in cm/s.

| Arguments:

| x: -500-500

| y: -500-500

| z: -500-500

| speed: 10-100

| yaw: -360-360

| mid1: 1-8

| mid2: 1-8

|

| initiate_throw_takeoff(self)

| Allows you to take off by throwing your drone within 5 seconds of this command

|

| land(self)

| Automatic landing.

|

```
| move(self, direction: str, x: int)
|     Tello fly up, down, left, right, forward or back with distance x cm.
|     Users would normally call one of the move_x functions instead.
|     Arguments:
|         direction: up, down, left, right, forward or back
|         x: 20-500
|
| move_back(self, x: int)
|     Fly x cm backwards.
|     Arguments:
|         x: 20-500
|
| move_down(self, x: int)
|     Fly x cm down.
|     Arguments:
|         x: 20-500
|
| move_forward(self, x: int)
|     Fly x cm forward.
|     Arguments:
|         x: 20-500
|
| move_left(self, x: int)
|     Fly x cm left.
|     Arguments:
|         x: 20-500
|
| move_right(self, x: int)
|     Fly x cm right.
|     Arguments:
|         x: 20-500
|
| move_up(self, x: int)
|     Fly x cm up.
|     Arguments:
|         x: 20-500
|
| parse_state(state: str) -> Dict[str, Union[int, float, str]]
|     Parse a state line to a dictionary
|     Internal method, you normally wouldn't call this yourself.
```

```
|
| query_active(self) -> str
|     Get the active status
|     Returns:
|         str
|
| query_attitude(self) -> dict
|     Query IMU attitude data.
|     Using get_pitch, get_roll and get_yaw is usually faster.
|     Returns:
|         {'pitch': int, 'roll': int, 'yaw': int}
|
| query_barometer(self) -> int
|     Get barometer value (cm)
|     Using get_barometer is usually faster.
|     Returns:
|         int: 0-100
|
| query_battery(self) -> int
|     Get current battery percentage via a query command
|     Using get_battery is usually faster
|     Returns:
|         int: 0-100 in %
|
| query_distance_tof(self) -> float
|     Get distance value from TOF (cm)
|     Using get_distance_tof is usually faster.
|     Returns:
|         float: 30-1000
|
| query_flight_time(self) -> int
|     Query current fly time (s).
|     Using get_flight_time is usually faster.
|     Returns:
|         int: Seconds elapsed during flight.
|
| query_height(self) -> int
|     Get height in cm via a query command.
|     Using get_height is usually faster
|     Returns:
```

```
|        int: 0-3000
|
| query_sdk_version(self) -> str
|     Get SDK Version
|     Returns:
|         str: SDK Version
|
| query_serial_number(self) -> str
|     Get Serial Number
|     Returns:
|         str: Serial Number
|
| query_speed(self) -> int
|     Query speed setting (cm/s)
|     Returns:
|         int: 1-100
|
| query_temperature(self) -> int
|     Query temperature (°C).
|     Using get_temperature is usually faster.
|     Returns:
|         int: 0-90
|
| query_wifi_signal_noise_ratio(self) -> str
|     Get Wi-Fi SNR
|     Returns:
|         str: snr
|
| raise_result_error(self, command: str, response: str) -> bool
|     Used to reaise an error after an unsuccessful command
|     Internal method, you normally wouldn't call this yourself.
|
| reboot(self)
|     Reboots the drone
|
| rotate_clockwise(self, x: int)
|     Rotate x degree clockwise.
|     Arguments:
|         x: 1-360
|
```

| rotate_counter_clockwise(self, x: int)

| Rotate x degree counter-clockwise.

| Arguments:

| x: 1-3600

|

| send_command_with_return(self, command: str, timeout: int = 7) -> str

| Send command to Tello and wait for its response.

| Internal method, you normally wouldn't call this yourself.

| Return:

| bool/str: str with response text on success, False when unsuccessfull.

|

| send_command_without_return(self, command: str)

| Send command to Tello without expecting a response.

| Internal method, you normally wouldn't call this yourself.

|

| send_control_command(self, command: str, timeout: int = 7) -> bool

| Send control command to Tello and wait for its response.

| Internal method, you normally wouldn't call this yourself.

|

| send_expansion_command(self, expansion_cmd: str)

| Sends a command to the ESP32 expansion board connected to a Tello Talent

| Use e.g. tello.send_expansion_command("led 255 0 0") to turn the top led red.

|

| send_keepalive(self)

| Send a keepalive packet to prevent the drone from landing after 15s

|

| send_rc_control(self, left_right_velocity: int, forward_backward_velocity: int, up_down_velocity: int, yaw_velocity: int)

| Send RC control via four channels. Command is sent every self.TIME_BTW_RC_CONTROL_COMMANDS seconds.

| Arguments:

| left_right_velocity: -100~100 (left/right)

| forward_backward_velocity: -100~100 (forward/backward)

| up_down_velocity: -100~100 (up/down)

| yaw_velocity: -100~100 (yaw)

|

| send_read_command(self, command: str) -> str

| Send given command to Tello and wait for its response.

| Internal method, you normally wouldn't call this yourself.

|

| send_read_command_float(self, command: str) -> float

| Send given command to Tello and wait for its response.

| Parses the response to an integer

| Internal method, you normally wouldn't call this yourself.

|

| send_read_command_int(self, command: str) -> int

| Send given command to Tello and wait for its response.

| Parses the response to an integer

| Internal method, you normally wouldn't call this yourself.

|

| set_mission_pad_detection_direction(self, x)

| Set mission pad detection direction. enable_mission_pads needs to be

| called first. When detecting both directions detecting frequency is 10Hz,

| otherwise the detection frequency is 20Hz.

| Arguments:

| x: 0 downwards only, 1 forwards only, 2 both directions

|

| set_network_ports(self, state_packet_port: int, video_stream_port: int)

| Sets the ports for state packets and video streaming

| While you can use this command to reconfigure the Tello this library currently does not support

| non-default ports (TODO!)

|

| set_speed(self, x: int)

| Set speed to x cm/s.

| Arguments:

| x: 10-100

|

| set_video_bitrate(self, bitrate: int)

| Sets the bitrate of the video stream

| Use one of the following for the bitrate argument:

| Tello.BITRATE_AUTO

| Tello.BITRATE_1MBPS

| Tello.BITRATE_2MBPS

| Tello.BITRATE_3MBPS

| Tello.BITRATE_4MBPS

| Tello.BITRATE_5MBPS

|

| set_video_direction(self, direction: int)

```
|     Selects one of the two cameras for video streaming
|     The forward camera is the regular 1080x720 color camera
|     The downward camera is a grey-only 320x240 IR-sensitive camera
|     Use one of the following for the direction argument:
|         Tello.CAMERA_FORWARD
|         Tello.CAMERA_DOWNWARD
|
| set_video_fps(self, fps: str)
|     Sets the frames per second of the video stream
|     Use one of the following for the fps argument:
|         Tello.FPS_5
|         Tello.FPS_15
|         Tello.FPS_30
|
| set_video_resolution(self, resolution: str)
|     Sets the resolution of the video stream
|     Use one of the following for the resolution argument:
|         Tello.RESOLUTION_480P
|         Tello.RESOLUTION_720P
|
| set_wifi_credentials(self, ssid: str, password: str)
|     Set the Wi-Fi SSID and password. The Tello will reboot afterwords.
|
| streamoff(self)
|     Turn off video streaming.
|
| streamon(self)
|     Turn on video streaming. Use `tello.get_frame_read` afterwards.
|     Video Streaming is supported on all tellos when in AP mode (i.e.
|     when your computer is connected to Tello-XXXXXX WiFi ntwork).
|     Currently Tello EDUs do not support video streaming while connected
|     to a WiFi-network.
|
|     !!! Note:
|         If the response is 'Unknown command' you have to update the Tello
|         firmware. This can be done using the official Tello app.
|
| takeoff(self)
|     Automatic takeoff.
|
```

| turn_motor_off(self)

| Turns off the motor cooling mode

|

| turn_motor_on(self)

| Turn on motors without flying (mainly for cooling)

|

| udp_response_receiver()

| Setup drone UDP receiver. This method listens for responses of Tello.

| Must be run from a background thread in order to not block the main thread.

| Internal method, you normally wouldn't call this yourself.

|

| udp_state_receiver()

| Setup state UDP receiver. This method listens for state information from

| Tello. Must be run from a background thread in order to not block

| the main thread.

| Internal method, you normally wouldn't call this yourself.

|

| ----------

| Data descriptors defined here:

|

| __dict__

| dictionary for instance variables (if defined)

|

| __weakref__

| list of weak references to the object (if defined)

|

| ----------

| Data and other attributes defined here:

|

| BITRATE_1MBPS = 1

|

| BITRATE_2MBPS = 2

|

| BITRATE_3MBPS = 3

|

| BITRATE_4MBPS = 4

|

| BITRATE_5MBPS = 5

|

| BITRATE_AUTO = 0

|

| CAMERA_DOWNWARD = 1

|

| CAMERA_FORWARD = 0

|

| CONTROL_UDP_PORT = 8889

|

| FLOAT_STATE_FIELDS = ('baro', 'agx', 'agy', 'agz')

|

| FORMATTER = <logging.Formatter object>

|

| FPS_15 = 'middle'

|

| FPS_30 = 'high'

|

| FPS_5 = 'low'

|

| FRAME_GRAB_TIMEOUT = 3

|

| HANDLER = <StreamHandler <stderr> (NOTSET)>

|

| INT_STATE_FIELDS = ('mid', 'x', 'y', 'z', 'pitch', 'roll', 'yaw', 'vgx...

|

| LOGGER = <Logger djitellopy (INFO)>

|

| RESOLUTION_480P = 'low'

|

| RESOLUTION_720P = 'high'

|

| RESPONSE_TIMEOUT = 7

|

| RETRY_COUNT = 3

|

| STATE_UDP_PORT = 8890

|

| TAKEOFF_TIMEOUT = 20

|

| TELLO_IP = '192.168.10.1'

|

```
 |  TIME_BTW_COMMANDS = 0.1
 |
 |  TIME_BTW_RC_CONTROL_COMMANDS = 0.001
 |
 |  VS_UDP_IP = '0.0.0.0'
 |
 |  VS_UDP_PORT = 11111
 |
 |  __annotations__ = {'background_frame_read':
typing.Optional[ForwardRef...
 |
 |  background_frame_read = None
 |
 |  cap = None
 |
 |  is_flying = False
 |
 |  state_field_converters = {'agx': <class 'float'>, 'agy': <class 'float...
 |
 |  stream_on = False
```

1_tello soket açma

```
import threading
import socket
import sys
import time
import platform

host=""
port=9000
locaddr=(host,port)

#create UDP socket
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind(locaddr)

tello_address=("192.168.10.1","8889")

sock.sendto(b'command',locaddr)
sock.sendto(b"takeoff", locaddr)
sock.sendto(b"left 25", locaddr)
sock.sendto(b"backward 25", locaddr)
sock.sendto(b"right 25", locaddr)
sock.sendto(b"forward 25", locaddr)

sock.sendto(b"battery?", locaddr)

response,ip=socket.recvform(1024)
print(response)

sock.sendto(b"land", locaddr)
sock.close()
```

2_tello_bilgi_cekme

```
import time, cv2
from threading import Thread
from djitellopy import Tello

tello = Tello()

tello.connect()
#kaldırmadan motor çalışsın
tello.turn_motor_on()

#barometre bilgisini al
print("Barometre bilgisi", tello.get_barometer())

#5 sn motor çalışsın
time.sleep(5)
print("Speed bilgisi", tello.get_speed_x())
print("Battery bilgisi", tello.get_battery())
print("Flight time bilgisi", tello.get_flight_time())
print("Snr signal noise ratio bilgisi", tello.query_wifi_signal_noise_ratio())
print("TOF time of flight distance in cm uzaklık bilgisi",
tello.get_distance_tof())

#kaldırmadan motor kapatsın
tello.turn_motor_off()
```

3_tello- ilk kalkış

```
from djitellopy import Tello
tello = Tello()

tello.connect()
tello.takeoff()

#tello.move_left(100)
tello.rotate_clockwise(100)
tello.move_forward(100)
"""
tello.move_back(30)
tello.flip_back()
tello.flip_right()
"""

tello.land()
```

4_tello Picture

```
import cv2
from djitellopy import Tello

tello = Tello()
tello.connect()

tello.streamon()
frame_read = tello.get_frame_read()

tello.takeoff()

cv2.imwrite("picture.png", frame_read.frame)

tello.land()
```

5_tello video

```
import time, cv2
from threading import Thread
from djitellopy import Tello

tello = Tello()
tello.connect()
tello.set_video_direction(Tello.CAMERA_FORWARD)
#tello.set_video_direction(Tello.CAMERA_DOWNWARD)

keepRecording = True
tello.streamon()
frame_read = tello.get_frame_read()

def videoRecorder():
    # create a VideoWrite object, recoring to ./video.avi
    height, width, _ = frame_read.frame.shape
    video = cv2.VideoWriter('video.avi', cv2.VideoWriter_fourcc(*'XVID'), 30,
(width, height))
    while keepRecording:
        video.write(frame_read.frame)
        time.sleep(1 / 30)

    video.release()

# we need to run the recorder in a seperate thread, otherwise blocking
options-  would prevent frames from getting added to the video

recorder = Thread(target=videoRecorder)
recorder.start()

print(tello.get_battery())
tello.takeoff()
tello.move_up(30)
tello.rotate_counter_clockwise(360)
tello.land()

keepRecording = False
recorder.join()
```

6_panorama

panoramaModule

```
#Module with individual panorama types defined. You can just import it and use hovever you like
#
#It will save photos from Tello inside folder that's in. You can change this by changing path inside every function.
from djitellopy import Tello
import cv2
import time

global img


def panorama_full_clockwise(tello_name):
    tello = tello_name
    tello.streamoff()
    tello.streamon()

    for i in range(4):
        img = tello.get_frame_read().frame
        cv2.imwrite(f'Panorama-full-clockwise_{time.time()}.jpg', img)
        time.sleep(1)
        tello.rotate_clockwise(80)

    img = tello.get_frame_read().frame
    cv2.imwrite(f'Panorama-full-clockwise_{time.time()}.jpg', img)
    time.sleep(1)
    tello.rotate_clockwise(40)

    tello.streamoff()


def panorama_half_clockwise(tello_name):
    tello = tello_name
    tello.streamoff()
    tello.streamon()

    tello.rotate_counter_clockwise(90)

    for i in range(3):
```

```
        img = tello.get_frame_read().frame
        cv2.imwrite(f'Panorama-half-clockwise_{time.time()}.jpg', img)
        time.sleep(1)
        tello.rotate_clockwise(60)

    img = tello.get_frame_read().frame
    cv2.imwrite(f'Panorama-half-clockwise_{time.time()}.jpg', img)
    time.sleep(1)
    tello.rotate_counter_clockwise(90)

    tello.streamoff()


def panorama_full_counter_clockwise(tello_name):
    tello = tello_name
    tello.streamoff()
    tello.streamon()

    for i in range(4):
        img = tello.get_frame_read().frame
        cv2.imwrite(f'Panorama-full-counter-clockwise_{time.time()}.jpg', img)
        time.sleep(1)
        tello.rotate_counter_clockwise(80)

    img = tello.get_frame_read().frame
    cv2.imwrite(f'/Panorama-full-counter-clockwise_{time.time()}.jpg', img)
    time.sleep(1)
    tello.rotate_counter_clockwise(40)

    tello.streamoff()


def panorama_half_counter_clockwise(tello_name):
    tello = tello_name
    tello.streamoff()
    tello.streamon()

    tello.rotate_clockwise(90)

    for i in range(3):
```

```
    img = tello.get_frame_read().frame
    cv2.imwrite(f'Panorama-half-counter-clockwise_{time.time()}.jpg', img)
    time.sleep(1)
    tello.rotate_counter_clockwise(60)

img = tello.get_frame_read().frame
cv2.imwrite(f'Panorama_half_counter_clockwise-{time.time()}.jpg', img)
time.sleep(1)
tello.rotate_clockwise(90)

tello.streamoff()
```

tello panorama

```
#Simply import of "panoramaModule.py" and you can use each function by calling it with name of the drone inside arguments.
from djitellopy import Tello
import cv2
import time
import panoramaModule


tello = Tello()
tello.connect()

print(tello.get_battery())

tello.takeoff()
tello.move_up(100)
panoramaModule.panorama_half_clockwise(tello)
tello.land()
```

7_tello tuş kontrol

```
# simple example demonstrating how to control a Tello using your keyboard.
# For a more fully featured example see manual-control-pygame.py
#
# Use W, A, S, D for moving, E, Q for rotating and R, F for going up and down.
# When starting the script the Tello will takeoff, pressing ESC makes it land
#  and the script exit.

from djitellopy import Tello
import cv2, math, time

tello = Tello()
tello.connect()
#tello.set_video_direction(Tello.CAMERA_FORWARD)
tello.streamon()
frame_read = tello.get_frame_read()

#tello.takeoff()
print("Komutlar: l : yere in ve kapat \n \
      w: ileri \n \
      s: geri \n \
      a: sola \n \
      d: sağa \n \
      e: döndür \n \
      q: ters yöne döndür \n \
      r: yukarı \n \
      f: aşağı \n \
      t: takla \n \ ")

while True:
    # In reality you want to display frames in a seperate thread. Otherwise
    #  they will freeze while the drone moves.
    img = frame_read.frame
    cv2.imshow("drone", img)
    key=input("Komut girin ve enter'a basın 'l,w,s,a,d,e,q,r,f':  ")
#    key = cv2.waitKey(1) and 0xff
    if key == "l": # land
        print("l ile yere in ve kapan")
        break
```

```
elif key == 'w':
    print("w ile 30 cm ileri")
    tello.move_forward(30)
elif key == 's':
    print("s ile 30 cm geri")
    tello.move_back(30)
elif key == 'a':
    print("a ile 30 cm sola")
    tello.move_left(30)
elif key == 'd':
    print("d ile 30 cm sağa")
    tello.move_right(30)
elif key == 'e':
    print("e ile 30  cm rotate")
    tello.rotate_clockwise(30)
elif key == 'q':
    print("q ile 30  cm rotate")
    tello.rotate_counter_clockwise(30)
elif key == 'r':
    print("r ile 30 cm yukarı")
    tello.move_up(30)
elif key == 'f':
    print("f ile 30 cm aşağı")
    tello.move_down(30)
elif key == 't':
    print("t ile takla")
    tello.flip("b")

#tello.land()
```

8_tello mission pads

```
from djitellopy import Tello


# create and connect
tello = Tello()
tello.connect()
print(tello.get_battery())


# configure drone


tello.set_video_direction(Tello.CAMERA_DOWNWARD)


tello.enable_mission_pads()
tello.set_mission_pad_detection_direction(2)  # forward detection only
#tello.takeoff()


pad = tello.get_mission_pad_id()
print("padi gördüm",pad,"x:",tello.get_mission_pad_distance_x(),"y:",tello.get_mission_pad_distance_y(),"z:",tello.get_mission_pad_distance_z())


# detect and react to pads until we see pad #1
n=0
while pad!=1:
   print("padi gördüm",pad,"x:",tello.get_mission_pad_distance_x(),"y:",tello.get_mission_pad_distance_y(),"z:",tello.get_mission_pad_distance_z())
   if pad!=-1:
      # tello.move_back(30)
      # tello.rotate_clockwise(90)
      pass
   n+=1
   print("dönüyorum",n)
   pad = tello.get_mission_pad_id()
"""
while pad != 1:
   if pad == 2:
      print("padi gördüm",pad,"x:",tello.get_mission_pad_distance_x(),"y:",tello.get_mission_pad_distance_y(),"z:",tello.get_mission_pad_distance_z())
      tello.move_back(30)
      tello.rotate_clockwise(90)
```

```
        print("burdayım", pad)


    n+=1
    print("dönüyorum",n)
    pad = tello.get_mission_pad_id()
    print("padi
gördüm",pad,"x:",tello.get_mission_pad_distance_x(),"y:",tello.get_mission_
pad_distance_y(),"z:",tello.get_mission_pad_distance_z())
"""
"""
    if pad == 4:
        tello.move_up(30)
        tello.flip_forward()
        print("burdayım 2")
"""


# graceful termination
tello.disable_mission_pads()
#tello.land()
print("tamam 1 i gördüm")
tello.end()
```