DJITelloPy

This documentation is the API reference of the DJITelloPy Library.

For more information on the project please see the readme on github.

API

Currently the library contains the following classes:

- [Tello][tello] for controlling a single tello drone.
- [Swarm][swarm] for controlling multiple Tello EDUs in parallel.

Example Code

Please see the example directory on github.

Installation

```
pip install djitellopy
```

For Linux distributions with both python2 and python3 (e.g. Debian, Ubuntu, ...) you need to run

```
pip3 install djitellopy
```

v: latest

v: latest

Swarm

Swarm library for controlling multiple Tellos simultaneously

`__getattr__(self, attr)` special

Call a standard tello function in parallel on all tellos.

```
swarm.command()
swarm.takeoff()
swarm.move_up(50)
```

Source code in `djitellopy/swarm.py`

~~~
def __getattr__(self, attr):
    """Call a standard tello function in parallel on all tellos.

    ```python
    swarm.command()
    swarm.takeoff()
    swarm.move_up(50)
    ```
    """
    def callAll(*args, **kwargs):
        self.parallel(lambda i, tello: getattr(tello, attr)(*args, **kwargs))

    return callAll
~~~

v: latest

`__init__(self, tellos)` special

Initialize a TelloSwarm instance

Parameters:

Name	Type	Description	Default
`tellos`	`List[djitellopy.tello.Tello]`	list of [Tello][tello] instances	*required*

Source code in `djitellopy/swarm.py`

```
def __init__(self, tellos: List[Tello]):
    """Initialize a TelloSwarm instance

    Arguments:
        tellos: list of [Tello][tello] instances
    """
    self.tellos = tellos
    self.barrier = Barrier(len(tellos))
    self.funcBarrier = Barrier(len(tellos) + 1)
    self.funcQueues = [Queue() for tello in tellos]

    def worker(i):
        queue = self.funcQueues[i]
        tello = self.tellos[i]

        while True:
            func = queue.get()
            self.funcBarrier.wait()
            func(i, tello)
            self.funcBarrier.wait()

    self.threads = []
    for i, _ in enumerate(tellos):
        thread = Thread(target=worker, daemon=True, args=(i,))
        thread.start()
        self.threads.append(thread)
```

v: latest

`__iter__(self)` special

Iterate over all drones in the swarm.

```
for tello in swarm:
    print(tello.get_battery())
```

Source code in `djitellopy/swarm.py`

~~~
def __iter__(self):
    """Iterate over all drones in the swarm.

    ```python
    for tello in swarm:
        print(tello.get_battery())
    ~~~
    """
    return iter(self.tellos)
```

`__len__(self)` special

Return the amount of tellos in the swarm

```
print("Tello count: {}".format(len(swarm)))
```

Source code in `djitellopy/swarm.py`

```
def __len__(self):
    """Return the amount of tellos in the swarm

    ```python
    print("Tello count: {}".format(len(swarm)))
    ```
    """
    return len(self.tellos)
```

v: latest
```

### `fromFile(path)`

Create TelloSwarm from file. The file should contain one IP address per line.

**Parameters:**

| Name | Type | Description | Default |
|---|---|---|---|
| `path` | `str` | path to the file | *required* |

**Source code in** `djitellopy/swarm.py`

```
@staticmethod
def fromFile(path: str):
    """Create TelloSwarm from file. The file should contain one IP address per line.

    Arguments:
        path: path to the file
    """
    with open(path, 'r') as fd:
        ips = fd.readlines()

    return TelloSwarm.fromIps(ips)
```

`fromIps(ips)`

Create TelloSwarm from a list of IP addresses.

Parameters:

v: latest

Name	Type	Description	Default
`ips`	`list`	list of IP Addresses	*required*

Source code in `djitellopy/swarm.py`

```
@staticmethod
def fromIps(ips: list):
    """Create TelloSwarm from a list of IP addresses.

    Arguments:
        ips: list of IP Addresses
    """
    if not ips:
        raise ValueError("No ips provided")

    tellos = []
    for ip in ips:
        tellos.append(Tello(ip.strip()))

    return TelloSwarm(tellos)
```

`parallel(self, func)`

Call `func` for each tello in parallel. The function retrieves two arguments: The index `i` of the current drone and `tello` the current [Tello][tello] instance.

You can use `swarm.sync()` for syncing between threads.

```
swarm.parallel(lambda i, tello: tello.move_up(50 + i * 10))
```

v: latest

Source code in `djitellopy/swarm.py`

~~~
def parallel(self, func: Callable[[int, Tello], None]):
    """Call `func` for each tello in parallel. The function retrieves
    two arguments: The index `i` of the current drone and `tello` the
    current [Tello][tello] instance.

    You can use `swarm.sync()` for syncing between threads.

    ```python
    swarm.parallel(lambda i, tello: tello.move_up(50 + i * 10))
    ```
    """

    for queue in self.funcQueues:
        queue.put(func)

    self.funcBarrier.wait()
    self.funcBarrier.wait()
~~~

`sequential(self, func)`

Call `func` for each tello sequentially. The function retrieves two arguments: The index `i` of the current drone and `tello` the current [Tello][tello] instance.

```
swarm.parallel(lambda i, tello: tello.land())
```

v: latest

Source code in `djitellopy/swarm.py`

~~~
def sequential(self, func: Callable[[int, Tello], None]):
    """Call `func` for each tello sequentially. The function retrieves
    two arguments: The index `i` of the current drone and `tello` the
    current [Tello][tello] instance.

    ```python
    swarm.parallel(lambda i, tello: tello.land())
    ```
    """

    for i, tello in enumerate(self.tellos):
        func(i, tello)
~~~

`sync(self, timeout=None)`

Sync parallel tello threads. The code continues when all threads have called `swarm.sync`.

```
def doStuff(i, tello):
    tello.move_up(50 + i * 10)
    swarm.sync()

    if i == 2:
        tello.flip_back()
    # make all other drones wait for one to complete its flip
    swarm.sync()

swarm.parallel(doStuff)
```

v: latest

Source code in `djitellopy/swarm.py`

~~~python
def sync(self, timeout: float = None):
    """Sync parallel tello threads. The code continues when all threads
    have called `swarm.sync`.

    ```python
    def doStuff(i, tello):
        tello.move_up(50 + i * 10)
        swarm.sync()

        if i == 2:
            tello.flip_back()
        # make all other drones wait for one to complete its flip
        swarm.sync()

    swarm.parallel(doStuff)
    ~~~
    """
    return self.barrier.wait(timeout)
```

v: latest
```

Tello

Python wrapper to interact with the Ryze Tello drone using the official Tello api. Tello API documentation: 1.3, 2.0 with EDU-only commands

`connect(self, wait_for_state=True)`

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

Source code in `djitellopy/tello.py`

```
def connect(self, wait_for_state=True):
    """Enter SDK mode. Call this before any of the control functions.
    """
    self.send_control_command("command")

    if wait_for_state:
        REPS = 20
        for i in range(REPS):
            if self.get_current_state():
                t = i / REPS  # in seconds
                Tello.LOGGER.debug("'.connect()' received first state packet after {} seconds".format(t))
                break
            time.sleep(1 / REPS)

        if not self.get_current_state():
            raise Exception('Did not receive a state packet from the Tello')
```

`connect_to_wifi(self, ssid, password)`

v: latest

Connects to the Wi-Fi with SSID and password. After this command the tello will reboot. Only works with Tello EDUs.

Source code in `djitellopy/tello.py`

```
def connect_to_wifi(self, ssid, password):
    """Connects to the Wi-Fi with SSID and password.
    After this command the tello will reboot.
    Only works with Tello EDUs.
    """
    cmd = 'ap {} {}'.format(ssid, password)
    self.send_command_without_return(cmd)
```

`curve_xyz_speed(self, x1, y1, z1, x2, y2, z2, speed)`

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.

Parameters:

Name	Type	Description	Default
`x1`	`int`	-500-500	*required*
`x2`	`int`	-500-500	*required*
`y1`	`int`	-500-500	*required*
`y2`	`int`	-500-500	*required*

v: latest

Name	Type	Description	Default
`z1`	`int`	-500-500	*required*
`z2`	`int`	-500-500	*required*
`speed`	`int`	10-60	*required*

Source code in `djitellopy/tello.py`

```
def 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
    """
    cmd = 'curve {} {} {} {} {} {} {}'.format(x1, y1, z1, x2, y2, z2, speed)
    self.send_control_command(cmd)
```

`curve_xyz_speed_mid(self, x1, y1, z1, x2, y2, z2, speed, mid)`

v: latest

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.

Parameters:

Name	Type	Description	Default
`x1`	`int`	-500-500	*required*
`y1`	`int`	-500-500	*required*
`z1`	`int`	-500-500	*required*
`x2`	`int`	-500-500	*required*
`y2`	`int`	-500-500	*required*
`z2`	`int`	-500-500	*required*
`speed`	`int`	10-60	*required*
`mid`	`int`	1-8	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    cmd = 'curve {} {} {} {} {} {} {} m{}'.format(x1, y1, z1, x2, y2, z2, speed, mid)
    self.send_control_command(cmd)
```

`disable_mission_pads(self)`

Disable mission pad detection

Source code in `djitellopy/tello.py`

```
def disable_mission_pads(self):
    """Disable mission pad detection
    """
    self.send_control_command("moff")
```

v: latest

`emergency(self)`

Stop all motors immediately.

Source code in `djitellopy/tello.py`

```
def emergency(self):
    """Stop all motors immediately.
    """
    self.send_control_command("emergency")
```

`enable_mission_pads(self)`

Enable mission pad detection

Source code in `djitellopy/tello.py`

```
def enable_mission_pads(self):
    """Enable mission pad detection
    """
    self.send_control_command("mon")
```

`end(self)`

Call this method when you want to end the tello object

v: latest

Source code in `djitellopy/tello.py`

```
def end(self):
    """Call this method when you want to end the tello object
    """
    if self.is_flying:
        self.land()
    if self.stream_on:
        self.streamoff()
    if self.background_frame_read is not None:
        self.background_frame_read.stop()
    if self.cap is not None:
        self.cap.release()

    host = self.address[0]
    if host in drones:
        del drones[host]
```

`flip(self, direction)`

Do a flip maneuver. Users would normally call one of the flip_x functions instead.

Parameters:

Name	Type	Description	Default
`direction`	`str`	l (left), r (right), f (forward) or b (back)	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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)
    """
    self.send_control_command("flip {}".format(direction))
```

`flip_back(self)`

Flip backwards.

Source code in `djitellopy/tello.py`

```
def flip_back(self):
    """Flip backwards.
    """
    self.flip("b")
```

`flip_forward(self)`

Flip forward.

Source code in `djitellopy/tello.py`

```
def flip_forward(self):
    """Flip forward.
    """
    self.flip("f")
```

v: latest

`flip_left(self)`

Flip to the left.

Source code in `djitellopy/tello.py`

```
def flip_left(self):
    """Flip to the left.
    """
    self.flip("l")
```

`flip_right(self)`

Flip to the right.

Source code in `djitellopy/tello.py`

```
def flip_right(self):
    """Flip to the right.
    """
    self.flip("r")
```

`get_acceleration_x(self)`

X-Axis Acceleration

Returns:

v: latest

Type	Description
`float`	float: acceleration

Source code in `djitellopy/tello.py`

```
def get_acceleration_x(self) -> float:
    """X-Axis Acceleration
    Returns:
        float: acceleration
    """
    return self.get_state_field('agx')
```

`get_acceleration_y(self)`

Y-Axis Acceleration

Returns:

Type	Description
`float`	float: acceleration

v: latest

Source code in `djitellopy/tello.py`

```
def get_acceleration_y(self) -> float:
    """Y-Axis Acceleration
    Returns:
        float: acceleration
    """
    return self.get_state_field('agy')
```

`get_acceleration_z(self)`

Z-Axis Acceleration

Returns:

Type	Description
`float`	float: acceleration

Source code in `djitellopy/tello.py`

```
def get_acceleration_z(self) -> float:
    """Z-Axis Acceleration
    Returns:
        float: acceleration
    """
    return self.get_state_field('agz')
```

`get_barometer(self)`

v: latest

Get current barometer measurement in cm This resembles the absolute height. See https://en.wikipedia.org/wiki/Altimeter

Returns:

Type	Description
`int`	int: barometer measurement in cm

Source code in `djitellopy/tello.py`

```
def 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
    """
    return self.get_state_field('baro') * 100
```

`get_battery(self)`

Get current battery percentage

Returns:

Type	Description
`int`	int: 0-100

v: latest

Source code in `djitellopy/tello.py`

```
def get_battery(self) -> int:
    """Get current battery percentage
    Returns:
        int: 0-100
    """
    return self.get_state_field('bat')
```

`get_current_state(self)`

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.

Source code in `djitellopy/tello.py`

```
def 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.
    """
    return self.get_own_udp_object()['state']
```

`get_distance_tof(self)`

Get current distance value from TOF in cm

Returns:

Type	Description
`int`	int: TOF distance in cm

v: latest

Source code in `djitellopy/tello.py`

```
def get_distance_tof(self) -> int:
    """Get current distance value from TOF in cm
    Returns:
        int: TOF distance in cm
    """
    return self.get_state_field('tof')
```

`get_flight_time(self)`

Get the time the motors have been active in seconds

Returns:

Type	Description
`int`	int: flight time in s

Source code in `djitellopy/tello.py`

```
def get_flight_time(self) -> int:
    """Get the time the motors have been active in seconds
    Returns:
        int: flight time in s
    """
    return self.get_state_field('time')
```

`get_frame_read(self)`

v: latest

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:

Type	Description
`BackgroundFrameRead`	BackgroundFrameRead

Source code in `djitellopy/tello.py`

```
def 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
    """
    if self.background_frame_read is None:
        address = self.get_udp_video_address()
        self.background_frame_read = BackgroundFrameRead(self, address)  # also sets self.cap
        self.background_frame_read.start()
    return self.background_frame_read
```

`get_height(self)`

Get current height in cm

Returns:

Type	Description
`int`	int: height in cm

v: latest

Source code in `djitellopy/tello.py`

```
def get_height(self) -> int:
    """Get current height in cm
    Returns:
        int: height in cm
    """
    return self.get_state_field('h')
```

`get_highest_temperature(self)`

Get highest temperature

Returns:

Type	Description
`int`	float: highest temperature (°C)

Source code in `djitellopy/tello.py`

```
def get_highest_temperature(self) -> int:
    """Get highest temperature
    Returns:
        float: highest temperature (°C)
    """
    return self.get_state_field('temph')
```

v: latest

`get_lowest_temperature(self)`

Get lowest temperature

Returns:

Type	Description
`int`	int: lowest temperature (°C)

Source code in `djitellopy/tello.py`

```
def get_lowest_temperature(self) -> int:
    """Get lowest temperature
    Returns:
        int: lowest temperature (°C)
    """
    return self.get_state_field('templ')
```

`get_mission_pad_distance_x(self)`

X distance to current mission pad Only available on Tello EDUs after calling enable_mission_pads

Returns:

Type	Description
`int`	int: distance in cm

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    return self.get_state_field('x')
```

get_mission_pad_distance_y(self)

Y distance to current mission pad Only available on Tello EDUs after calling enable_mission_pads

Returns:

Type	Description
`int`	int: distance in cm

Source code in `djitellopy/tello.py`

```
def 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
    """
    return self.get_state_field('y')
```

v: latest

get_mission_pad_distance_z(self)

Z distance to current mission pad Only available on Tello EDUs after calling enable_mission_pads

Returns:

Type	Description
`int`	int: distance in cm

Source code in `djitellopy/tello.py`

```
def 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
    """
    return self.get_state_field('z')
```

`get_mission_pad_id(self)`

Mission pad ID of the currently detected mission pad Only available on Tello EDUs after calling enable_mission_pads

Returns:

Type	Description
`int`	int: -1 if none is detected, else 1-8

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    return self.get_state_field('mid')
```

`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.

Source code in `djitellopy/tello.py`

```
def 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.
    """
    global drones

    host = self.address[0]
    return drones[host]
```

`get_pitch(self)`

Get pitch in degree

v: latest

Returns:

Type	Description
`int`	int: pitch in degree

Source code in `djitellopy/tello.py`

```
def get_pitch(self) -> int:
    """Get pitch in degree
    Returns:
        int: pitch in degree
    """
    return self.get_state_field('pitch')
```

`get_roll(self)`

Get roll in degree

Returns:

Type	Description
`int`	int: roll in degree

v: latest

Source code in `djitellopy/tello.py`

```
def get_roll(self) -> int:
    """Get roll in degree
    Returns:
        int: roll in degree
    """
    return self.get_state_field('roll')
```

`get_speed_x(self)`

X-Axis Speed

Returns:

Type	Description
`int`	int: speed

Source code in `djitellopy/tello.py`

```
def get_speed_x(self) -> int:
    """X-Axis Speed
    Returns:
        int: speed
    """
    return self.get_state_field('vgx')
```

`get_speed_y(self)`

v: latest

Y-Axis Speed

Returns:

Type	Description
`int`	int: speed

Source code in `djitellopy/tello.py`

```
def get_speed_y(self) -> int:
    """Y-Axis Speed
    Returns:
        int: speed
    """
    return self.get_state_field('vgy')
```

`get_speed_z(self)`

Z-Axis Speed

Returns:

Type	Description
`int`	int: speed

v: latest

Source code in `djitellopy/tello.py`

```
def get_speed_z(self) -> int:
    """Z-Axis Speed
    Returns:
        int: speed
    """
    return self.get_state_field('vgz')
```

`get_state_field(self, key)`

Get a specific sate field by name. Internal method, you normally wouldn't call this yourself.

Source code in `djitellopy/tello.py`

```
def get_state_field(self, key: str):
    """Get a specific sate field by name.
    Internal method, you normally wouldn't call this yourself.
    """
    state = self.get_current_state()

    if key in state:
        return state[key]
    else:
        raise Exception('Could not get state property: {}'.format(key))
```

`get_temperature(self)`

Get average temperature

Returns:

v: latest

Type	Description
`float`	float: average temperature (°C)

Source code in `djitellopy/tello.py`

```
def get_temperature(self) -> float:
    """Get average temperature
    Returns:
        float: average temperature (°C)
    """
    templ = self.get_lowest_temperature()
    temph = self.get_highest_temperature()
    return (templ + temph) / 2
```

`get_udp_video_address(self)`

Internal method, you normally wouldn't call this youself.

Source code in `djitellopy/tello.py`

```
def get_udp_video_address(self) -> str:
    """Internal method, you normally wouldn't call this youself.
    """
    address_schema = 'udp://@{ip}:{port}'  # + '?overrun_nonfatal=1&fifo_size=5000'
    address = address_schema.format(ip=self.VS_UDP_IP, port=self.VS_UDP_PORT)
    return address
```

v: latest

`get_video_capture(self)`

Get the VideoCapture object from the camera drone. Users usually want to use get_frame_read instead.

Returns:

Type	Description
	VideoCapture

Source code in `djitellopy/tello.py`

```
def get_video_capture(self):
    """Get the VideoCapture object from the camera drone.
    Users usually want to use get_frame_read instead.
    Returns:
        VideoCapture
    """

    if self.cap is None:
        self.cap = cv2.VideoCapture(self.get_udp_video_address())

    if not self.cap.isOpened():
        self.cap.open(self.get_udp_video_address())

    return self.cap
```

`get_yaw(self)`

Get yaw in degree

Returns:

Type	Description
`int`	int: yaw in degree

v: latest

Source code in `djitellopy/tello.py`

```
def get_yaw(self) -> int:
    """Get yaw in degree
    Returns:
        int: yaw in degree
    """
    return self.get_state_field('yaw')
```

`go_xyz_speed(self, x, y, z, speed)`

Fly to x y z relative to the current position. Speed defines the traveling speed in cm/s.

Parameters:

Name	Type	Description	Default
`x`	`int`	-500-500	*required*
`y`	`int`	-500-500	*required*
`z`	`int`	-500-500	*required*
`speed`	`int`	10-100	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    cmd = 'go {} {} {} {}'.format(x, y, z, speed)
    self.send_control_command(cmd)
```

`go_xyz_speed_mid(self, x, y, z, speed, mid)`

Fly to x y z relative to the mission pad with id mid. Speed defines the traveling speed in cm/s.

Parameters:

Name	Type	Description	Default
`x`	`int`	-500-500	*required*
`y`	`int`	-500-500	*required*
`z`	`int`	-500-500	*required*
`speed`	`int`	10-100	*required*
`mid`	`int`	1-8	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    cmd = 'go {} {} {} {} m{}'.format(x, y, z, speed, mid)
    self.send_control_command(cmd)
```

`go_xyz_speed_yaw_mid(self, x, y, z, speed, yaw, mid1, mid2)`

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.

Parameters:

Name	Type	Description	Default
`x`	`int`	-500-500	*required*
`y`	`int`	-500-500	*required*
`z`	`int`	-500-500	*required*
`speed`	`int`	10-100	*required*
`yaw`	`int`	-360-360	*required*

v: latest

Name	Type	Description	Default
`mid1`	`int`	1-8	*required*
`mid2`	`int`	1-8	*required*

Source code in `djitellopy/tello.py`

```
def 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
    """
    cmd = 'jump {} {} {} {} {} m{} m{}'.format(x, y, z, speed, yaw, mid1, mid2)
    self.send_control_command(cmd)
```

`land(self)`

Automatic landing.

v: latest

Source code in `djitellopy/tello.py`

```
def land(self):
    """Automatic landing.
    """
    self.send_control_command("land")
    self.is_flying = False
```

`move(self, direction, x)`

Tello fly up, down, left, right, forward or back with distance x cm. Users would normally call one of the move_x functions instead.

Parameters:

Name	Type	Description	Default
`direction`	`str`	up, down, left, right, forward or back	*required*
`x`	`int`	20-500	*required*

Source code in `djitellopy/tello.py`

```
def 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
    """
    self.send_control_command("{} {}".format(direction, x))
```

v: latest

`move_back(self, x)`

Fly x cm backwards.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

Source code in `djitellopy/tello.py`

```
def move_back(self, x: int):
    """Fly x cm backwards.
    Arguments:
        x: 20-500
    """
    self.move("back", x)
```

`move_down(self, x)`

Fly x cm down.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def move_down(self, x: int):
    """Fly x cm down.
    Arguments:
        x: 20-500
    """
    self.move("down", x)
```

`move_forward(self, x)`

Fly x cm forward.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

Source code in `djitellopy/tello.py`

```
def move_forward(self, x: int):
    """Fly x cm forward.
    Arguments:
        x: 20-500
    """
    self.move("forward", x)
```

`move_left(self, x)`

v: latest

Fly x cm left.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

Source code in `djitellopy/tello.py`

```
def move_left(self, x: int):
    """Fly x cm left.
    Arguments:
        x: 20-500
    """
    self.move("left", x)
```

`move_right(self, x)`

Fly x cm right.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def move_right(self, x: int):
    """Fly x cm right.
    Arguments:
        x: 20-500
    """
    self.move("right", x)
```

`move_up(self, x)`

Fly x cm up.

Parameters:

Name	Type	Description	Default
`x`	`int`	20-500	*required*

Source code in `djitellopy/tello.py`

```
def move_up(self, x: int):
    """Fly x cm up.
    Arguments:
        x: 20-500
    """
    self.move("up", x)
```

`parse_state(state)`

v: latest

Parse a state line to a dictionary Internal method, you normally wouldn't call this yourself.

Source code in `djitellopy/tello.py`

```
@staticmethod
def 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.
    """
    state = state.strip()
    Tello.LOGGER.debug('Raw state data: {}'.format(state))

    if state == 'ok':
        return {}

    state_dict = {}
    for field in state.split(';'):
        split = field.split(':')
        if len(split) < 2:
            continue

        key = split[0]
        value: Union[int, float, str] = split[1]

        if key in Tello.state_field_converters:
            num_type = Tello.state_field_converters[key]
            try:
                value = num_type(value)
            except ValueError as e:
                Tello.LOGGER.debug('Error parsing state value for {}: {} to {}'
                                   .format(key, value, num_type))
                Tello.LOGGER.error(e)
                continue

        state_dict[key] = value

    return state_dict
```

`query_attitude(self)`

v: latest

Query IMU attitude data. Using get_pitch, get_roll and get_yaw is usually faster.

Returns:

Type	Description
`dict`	{'pitch': int, 'roll': int, 'yaw': int}

Source code in `djitellopy/tello.py`

```
def 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}
    """
    response = self.send_read_command('attitude?')
    return Tello.parse_state(response)
```

`query_barometer(self)`

Get barometer value (cm) Using get_barometer is usually faster.

Returns:

Type	Description
`int`	int: 0-100

v: latest

Source code in `djitellopy/tello.py`

```
def query_barometer(self) -> int:
    """Get barometer value (cm)
    Using get_barometer is usually faster.
    Returns:
        int: 0-100
    """
    baro = self.send_read_command_int('baro?')
    return baro * 100
```

`query_battery(self)`

Get current battery percentage via a query command Using get_battery is usually faster

Returns:

Type	Description
`int`	int: 0-100 in %

Source code in `djitellopy/tello.py`

```
def query_battery(self) -> int:
    """Get current battery percentage via a query command
    Using get_battery is usually faster
    Returns:
        int: 0-100 in %
    """
    return self.send_read_command_int('battery?')
```

v: latest

`query_distance_tof(self)`

Get distance value from TOF (cm) Using get_distance_tof is usually faster.

Returns:

Type	Description
`float`	float: 30-1000

Source code in `djitellopy/tello.py`

```
def query_distance_tof(self) -> float:
    """Get distance value from TOF (cm)
    Using get_distance_tof is usually faster.
    Returns:
        float: 30-1000
    """
    # example response: 801mm
    tof = self.send_read_command('tof?')
    return int(tof[:-2]) / 10
```

`query_flight_time(self)`

Query current fly time (s). Using get_flight_time is usually faster.

Returns:

Type	Description
`int`	int: Seconds elapsed during flight.

v: latest

Type	Description

Source code in `djitellopy/tello.py`

```
def query_flight_time(self) -> int:
    """Query current fly time (s).
    Using get_flight_time is usually faster.
    Returns:
        int: Seconds elapsed during flight.
    """
    return self.send_read_command_int('time?')
```

`query_height(self)`

Get height in cm via a query command. Using get_height is usually faster

Returns:

Type	Description
`int`	int: 0-3000

Source code in `djitellopy/tello.py`

```
def query_height(self) -> int:
    """Get height in cm via a query command.
    Using get_height is usually faster
    Returns:
        int: 0-3000
    """
    return self.send_read_command_int('height?')
```

v: latest

`query_sdk_version(self)`

Get SDK Version

Returns:

Type	Description
`str`	str: SDK Version

Source code in `djitellopy/tello.py`

```
def query_sdk_version(self) -> str:
    """Get SDK Version
    Returns:
        str: SDK Version
    """
    return self.send_read_command('sdk?')
```

`query_serial_number(self)`

Get Serial Number

Returns:

Type	Description
`str`	str: Serial Number

v: latest

Source code in `djitellopy/tello.py`

```
def query_serial_number(self) -> str:
    """Get Serial Number
    Returns:
        str: Serial Number
    """
    return self.send_read_command('sn?')
```

`query_speed(self)`

Query speed setting (cm/s)

Returns:

Type	Description
`int`	int: 1-100

Source code in `djitellopy/tello.py`

```
def query_speed(self) -> int:
    """Query speed setting (cm/s)
    Returns:
        int: 1-100
    """
    return self.send_read_command_int('speed?')
```

`query_temperature(self)`

v: latest

Query temperature (°C). Using get_temperature is usually faster.

Returns:

Type	Description
`int`	int: 0-90

Source code in `djitellopy/tello.py`

```
def query_temperature(self) -> int:
    """Query temperature (°C).
    Using get_temperature is usually faster.
    Returns:
        int: 0-90
    """
    return self.send_read_command_int('temp?')
```

`query_wifi_signal_noise_ratio(self)`

Get Wi-Fi SNR

Returns:

Type	Description
`str`	str: snr

v: latest

Source code in `djitellopy/tello.py`

```
def query_wifi_signal_noise_ratio(self) -> str:
    """Get Wi-Fi SNR
    Returns:
        str: snr
    """
    return self.send_read_command('wifi?')
```

`raise_result_error(self, command, response)`

Used to reaise an error after an unsuccessful command Internal method, you normally wouldn't call this yourself.

Source code in `djitellopy/tello.py`

```
def 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.
    """
    tries = 1 + self.retry_count
    raise Exception("Command '{}' was unsuccessful for {} tries. Latest response:\t'{}'"
                    .format(command, tries, response))
```

`rotate_clockwise(self, x)`

Rotate x degree clockwise.

Parameters:

Name	Type	Description	Default
`x`	`int`	1-360	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def rotate_clockwise(self, x: int):
    """Rotate x degree clockwise.
    Arguments:
        x: 1-360
    """
    self.send_control_command("cw {}".format(x))
```

`rotate_counter_clockwise(self, x)`

Rotate x degree counter-clockwise.

Parameters:

Name	Type	Description	Default
`x`	`int`	1-3600	*required*

Source code in `djitellopy/tello.py`

```
def rotate_counter_clockwise(self, x: int):
    """Rotate x degree counter-clockwise.
    Arguments:
        x: 1-3600
    """
    self.send_control_command("ccw {}".format(x))
```

`send_command_with_return(self, command, timeout=7)`

v: latest

Send command to Tello and wait for its response. Internal method, you normally wouldn't call this yourself.

Returns:

Type	Description
`str`	bool/str: str with response text on success, False when unsuccessfull.

v: latest

Source code in `djitellopy/tello.py`

```
def send_command_with_return(self, command: str, timeout: int = RESPONSE_TIMEOUT) -> 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.
    """
    # Commands very consecutive makes the drone not respond to them.
    # So wait at least self.TIME_BTW_COMMANDS seconds
    diff = time.time() - self.last_received_command_timestamp
    if diff < self.TIME_BTW_COMMANDS:
        self.LOGGER.debug('Waiting {} seconds to execute command: {}...'.format(diff, command))
        time.sleep(diff)

    self.LOGGER.info("Send command: '{}'".format(command))
    timestamp = time.time()

    client_socket.sendto(command.encode('utf-8'), self.address)

    responses = self.get_own_udp_object()['responses']

    while not responses:
        if time.time() - timestamp > timeout:
            message = "Aborting command '{}'. Did not receive a response after {} seconds".format(command, timeout)
            self.LOGGER.warning(message)
            return message
        time.sleep(0.1)  # Sleep during send command

    self.last_received_command_timestamp = time.time()

    first_response = responses.pop(0)  # first datum from socket
    try:
        response = first_response.decode("utf-8")
    except UnicodeDecodeError as e:
        self.LOGGER.error(e)
        return "response decode error"
    response = response.rstrip("\r\n")

    self.LOGGER.info("Response {}: '{}'".format(command, response))
    return response
```

v: latest

`send_command_without_return(self, command)`

Send command to Tello without expecting a response. Internal method, you normally wouldn't call this yourself.

Source code in `djitellopy/tello.py`

```
def send_command_without_return(self, command: str):
    """Send command to Tello without expecting a response.
    Internal method, you normally wouldn't call this yourself.
    """
    # Commands very consecutive makes the drone not respond to them. So wait at least self.TIME_BTW_COMMANDS seconds

    self.LOGGER.info("Send command (no response expected): '{}'".format(command))
    client_socket.sendto(command.encode('utf-8'), self.address)
```

`send_control_command(self, command, timeout=7)`

Send control command to Tello and wait for its response. Internal method, you normally wouldn't call this yourself.

v: latest

Source code in `djitellopy/tello.py`

```
def send_control_command(self, command: str, timeout: int = RESPONSE_TIMEOUT) -> bool:
    """Send control command to Tello and wait for its response.
    Internal method, you normally wouldn't call this yourself.
    """
    response = "max retries exceeded"
    for i in range(0, self.retry_count):
        response = self.send_command_with_return(command, timeout=timeout)

        if response.lower() == 'ok':
            return True

        self.LOGGER.debug("Command attempt #{} failed for command: '{}'".format(i, command))

    self.raise_result_error(command, response)
    return False # never reached
```

`send_rc_control(self, left_right_velocity, forward_backward_velocity, up_down_velocity, yaw_velocity)`

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

Parameters:

Name	Type	Description	Default
`left_right_velocity`	`int`	-100~100 (left/right)	*required*
`forward_backward_velocity`	`int`	-100~100 (forward/backward)	*required*
`up_down_velocity`	`int`	-100~100 (up/down)	*required*
`yaw_velocity`	`int`	-100~100 (yaw)	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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)
    """
    def clamp100(x: int) -> int:
        return max(-100, min(100, x))

    if time.time() - self.last_rc_control_timestamp > self.TIME_BTW_RC_CONTROL_COMMANDS:
        self.last_rc_control_timestamp = time.time()
        cmd = 'rc {} {} {} {}'.format(
            clamp100(left_right_velocity),
            clamp100(forward_backward_velocity),
            clamp100(up_down_velocity),
            clamp100(yaw_velocity)
        )
        self.send_command_without_return(cmd)
```

`send_read_command(self, command)`

Send given command to Tello and wait for its response. Internal method, you normally wouldn't call this yourself.

v: latest

Source code in `djitellopy/tello.py`

```
def 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.
    """

    response = self.send_command_with_return(command)

    try:
        response = str(response)
    except TypeError as e:
        self.LOGGER.error(e)

    if any(word in response for word in ('error', 'ERROR', 'False')):
        self.raise_result_error(command, response)
        return "Error: this code should never be reached"

    return response
```

`send_read_command_float(self, command)`

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.

Source code in `djitellopy/tello.py`

```
def 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.
    """
    response = self.send_read_command(command)
    return float(response)
```

v: latest

`send_read_command_int(self, command)`

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.

Source code in `djitellopy/tello.py`

```
def 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.
    """
    response = self.send_read_command(command)
    return int(response)
```

`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.

Parameters:

Name	Type	Description	Default
`x`		0 downwards only, 1 forwards only, 2 both directions	*required*

v: latest

Source code in `djitellopy/tello.py`

```
def 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
    """
    self.send_control_command("mdirection {}".format(x))
```

`set_speed(self, x)`

Set speed to x cm/s.

Parameters:

Name	Type	Description	Default
`x`	`int`	10-100	*required*

Source code in `djitellopy/tello.py`

```
def set_speed(self, x: int):
    """Set speed to x cm/s.
    Arguments:
        x: 10-100
    """
    self.send_control_command("speed {}".format(x))
```

v: latest

`set_wifi_credentials(self, ssid, password)`

Set the Wi-Fi SSID and password. The Tello will reboot afterwords.

Source code in `djitellopy/tello.py`

```
def set_wifi_credentials(self, ssid, password):
    """Set the Wi-Fi SSID and password. The Tello will reboot afterwords.
    """
    cmd = 'wifi {} {}'.format(ssid, password)
    self.send_command_without_return(cmd)
```

`streamoff(self)`

Turn off video streaming.

Source code in `djitellopy/tello.py`

```
def streamoff(self):
    """Turn off video streaming.
    """
    self.send_control_command("streamoff")
    self.stream_on = False
```

`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.

v: latest

Source code in `djitellopy/tello.py`

```
def 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.
    """
    self.send_control_command("streamon")
    self.stream_on = True
```

`takeoff(self)`

Automatic takeoff.

Source code in `djitellopy/tello.py`

```
def takeoff(self):
    """Automatic takeoff.
    """
    # Something it takes a looooot of time to take off and return a succesful takeoff.
    # So we better wait. Otherwise, it would give us an error on the following calls.
    self.send_control_command("takeoff", timeout=Tello.TAKEOFF_TIMEOUT)
    self.is_flying = True
```

`udp_response_receiver()`

v: latest

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.

Source code in `djitellopy/tello.py`

```
@staticmethod
def 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.
    """
    while True:
        try:
            data, address = client_socket.recvfrom(1024)

            address = address[0]
            Tello.LOGGER.debug('Data received from {} at client_socket'.format(address))

            if address not in drones:
                continue

            drones[address]['responses'].append(data)

        except Exception as e:
            Tello.LOGGER.error(e)
            break
```

`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.

v: latest

Source code in `djitellopy/tello.py`

```
@staticmethod
def 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.
    """
    state_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    state_socket.bind(("", Tello.STATE_UDP_PORT))

    while True:
        try:
            data, address = state_socket.recvfrom(1024)

            address = address[0]
            Tello.LOGGER.debug('Data received from {} at state_socket'.format(address))

            if address not in drones:
                continue

            data = data.decode('ASCII')
            drones[address]['state'] = Tello.parse_state(data)

        except Exception as e:
            Tello.LOGGER.error(e)
            break
```

v: latest