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`

```
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