DronLink: Funciones de la clase Dron

Se describen en la tabla siguiente las funciones de la clase Dron en la versión actual de la librería DronLink. En muchas de ellas aparecen los tres parámetros requeridos para implementar la modalidad no bloqueante (`blocking, callback` y `params`).

```
def connect(self,
   connection string, baud,
   id=None, freq = 4,
   blocking=True, callback=None, params = None)
```

Conecta con el dron. Los parámetros `connection_string, baud` indican si hay que conectar con el simulador o con el dron real, y la velocidad de comunicación. Los strings de conexón habituales son:

Simulador: `'tcp:127.0.0.1:5763'` y velocidad de 115200.
Directamente al dron por radio de telemetría: `'com??'` y velocidad 57600
Con el dron a través de mavproxy: `'udp:127.0.0.1:14551'` y velocidad de 115200.

La conexión admite una identificador para el dron, que la librería añadirá como primer parámetro en todas las funcione callback. El parámetro `freq` indica la frecuencia con la que se van a enviar datos de telemetría. Por defecto se envían 4 paquetes de datos por segundo. Con frecuencias mayores puede conseguirse, por ejemplo, mayor fluidez en el movimiento del icono del dron sobre un mapa, aunque puede saturarse un poco el computador en el que se ejecuta la aplicación. La conexión suele ser rápida. Por eso es habitual usar el modo bloqueante al usar esta función.

```
def disconnect (self)
```

Desconecta el dron. Además, detiene el envío de datos de telemetría.

```
def arm(self, blocking=True, callback=None, params = None)
```

Arma el dron. El armado es rápido. Por eso es habitual usar el modo bloqueante al usar esta función.

```
def takeOff(self,
   aTargetAltitude,
   blocking=True, callback=None, params = None)
```

Despega el dron hasta alcanzar la altura indicada en el parámetro.

```
def changeNavSpeed (self, speed)
```

Cambia la velocidad de navegación.

```
def go(self, direction)
```

Hace que el dron navegue en la dirección indicada. Las opciones son: `'North', 'South', 'West', 'East', 'NorthWest', 'NorthEast', 'SouthWest', 'SouthEast', 'Stop', 'Forward', 'Back', 'Left', 'Right', 'Up', 'Down'.`

```
def getParams(self,
   parameters,
   blocking=True, callback=None)
```

Pide al dron el valor de los parámetros indicados. En el caso de que la llamada sea o bloqueante, ejecutará la función del callback pasándole como parámetro la lista de valores recibida (y el identificador del dron como primer parámetro en el caso de que el dron haya sido identificando en el momento de la conexión). Este ejemplo muestra el formato en el que debe pasarse la lista de parámetros y el formato de la respuesta.

```
parameters = [
    "RTL_ALT",
    "PILOT_SPEED_UP",
    "FENCE_ACTION"
]
result = dron.getParams(parameters)
print('Valores:',result)
```

El resultado podría ser:

```
Valores: [{'RTL_ALT': -1.0}, {'PILOT_SPEED_UP': 100.0}, {'FENCE_ACTION': 4.0}]
```

```
def setParams(self,
   parameters,
   blocking=True, callback=None, params = None)
```

Establece el valor de los parámetros que se le pasan en una lista. Un ejemplo de uso es este:

```
parameters =[
        {'ID': "FENCE_ENABLE", 'Value': 1},
        {'ID': "FENCE_ACTION", 'Value': 4}
]
dron.setParams(parameters)
```

```
def RTL (self, blocking=True, callback=None, params = None)
```

Ordena al dron que retorne a casa (Return to launch).

```
def Land (self, blocking=True, callback=None, params = None)
```

Ordena al dron que aterrice en el punto que está sobrevolando.

```
def send_telemetry_info(self, process_telemetry_info)
```

Pide al dron que envíe los datos de telemetría. Cuando el dron tiene un nuevo paquete de telemetría llama al callback y le pasa ese paquete (y el identificador del dron como primer parámetro en el caso de que el dron haya sido identificando en el momento de la conexión). El dron va a enviar tantos paquetes por segundo como indique el parámetro `freq` en el momento de la conexión. El ejemplo siguiente muestra el formato en que se reciben los datos de telemetría:

```
def process_telemetry_info(self, telemetry_info):
    print ('info: ', telemetry_info)
```

El resultado podría ser:

```
info:{'lat':41.276,'lon':1.988, 'alt': -0.016, 'groundSpeed': 0.120, 'heading':
355.88, 'state': 'connected'}
```

Los posibles estados en los que puede estar el dron son: `'connected'`, `'disconnected'`, `'arming'`, `'armed'`, `'takingOff'`, `'flying'`, `'returning'`, `'landing'`.

```
def stop_sending_telemetry_info(self)
```

Detiene el envío de datos de telemetría.

```
def goto(self,
   lat, lon, alt,
   blocking=True, callback=None, params=None)
```

Dirige al dron al punto geográfico indicado.

```
def change_altitude(self,
   alt,
   blocking=True, callback=None, params=None)
```

Cambia la altura a la que se encuentra el dron.

```
def send_local_telemetry_info(self, process_local_telemetry_info)
```

Pide al dron que envíe los datos de telemetría local. Cuando el dron tiene un nuevo paquete de telemetría llama al callback y le pasa ese paquete (y el identificador del dron como primer parámetro en el caso de que el dron haya sido identificando en el momento de la conexión). El dron va a enviar tantos paquetes por segundo como indique el parámetro `freq` en el momento de la conexión. El ejemplo siguiente muestra el formato en que se reciben los datos de telemetría:

```
def process_local_telemetry_info(self, telemetry_info):
    print ('info: ', telemetry_info)
```

El resultado podría ser:

```
info:{'posX':3 ,'posY':2.5, 'posZ': -3}
```

El valor `posX` es el desplazamiento en metros en el que se encuentra el dron en la dirección del Norte, respeto al home (o en la dirección de Sur si el valor es negativo). De manera similar, `posY` indica el desplazamiento en la dirección Este (u Oeste para valores negativos) y `posZ` el desplazamiento hacia abajo (o hacia arriba para valores negativos).

```
def stop_sending_local_telemetry_info(self)
```

Detiene el envío de datos de telemetría local.

```
def uploadMission(self,
   mission,
   blocking=True, callback=None, params = None)
```

Carga en el autopiloto la misión indicada, que debe especificarse en ese formato:

```
{
    "takeOffAlt": 5,
    "waypoints":
        [
            {
                'lat': 41.2763410,
                'lon': 1.9888285,
                'alt': 12
            },
            {
                'lat': 41.27623,
                'lon': 1.987,
                'alt': 14
            }
        ]

}
```

El dron despegará en la posición en la que esté y al llegar al último waypoint harán un RTL.

```
def getMission(self,
   blocking=True, callback=None)
```

Retorna la misión que está cargada en ese momento en el autopiloto (o None si no hay misión). En el caso de que la llamada sea no bloqueante llamará a la función callback pasándole como parámetro la misión. El formato en que retorna la misión es el indicado en la descripción del método `uploadMission`.

```
def executeMission(self,
   blocking=True, callback=None, params = None)
```

El dron ejecuta la última misión que se haya cargado.

```
def setScenario(self,
   scenario,
   blocking=True, callback=None, params = None)
```

El escenario se recibe en forma de lista. En cada posición hay un fence que representan áreas.
El primer elemento de la lista es un fence de inclusión, que representa el área de la que el dron no va a salir. El resto de elementos de la lista son fences de exclusión que representan obstáculos dentro del fence de inclusión, que el dron no puede sobrevolar. El escenario debe tener un fence de inclusión (solo uno y es el primer elemento de la lista) y un número variable de fences de exclusión, que puede ser 0.
Un fence (tanto de inclusión como de exclusión) puede ser de tipo 'polygon' o de tipo 'circle'. En el primer caso el fence se caracteriza por un número variable de waypoints (lat, lon). Deben ser al menos 3 puesto que representan los vértices del polígono. Si el fence es de tipo 'circle' debe especificarse las coordenadas (lat, lon) del centro del círculo y el radio en metros.
Un ejemplo de scenario en el formato correcto es este:

```
scenario = [
        {
            'type': 'polygon',
            'waypoints': [
                {'lat': 41.2764398, 'lon': 1.9882585},
                {'lat': 41.2761999, 'lon': 1.9883537},
                {'lat': 41.2763854, 'lon': 1.9890994},
                {'lat': 41.2766273, 'lon': 1.9889948}
            ]
        },
        {
            'type': 'polygon',
            'waypoints': [
                {'lat': 41.2764801, 'lon': 1.9886541},
                {'lat': 41.2764519, 'lon': 1.9889626},
                {'lat': 41.2763995, 'lon': 1.9887963},
            ]
        },
        {
            'type': 'polygon',
            'waypoints': [
                {'lat': 41.2764035, 'lon': 1.9883262},
                {'lat': 41.2762160, 'lon': 1.9883537},
                {'lat': 41.2762281, 'lon': 1.9884771}
            ]
        },
        {
            'type': 'circle',
            'radius': 2,
            'lat': 41.2763430,
            'lon': 1.9883953
        }
    ]
```

El escenario tiene 4 fences. El primero es el de inclusión, de tipo 'polygon'. Luego tiene 3 fences de exclusión que representan los obstáculos. Los dos primeros son de tipo 'polygon' y el tercero es de tipo 'circle'.

```
def getScenario(self,
   blocking=True, callback=None)
```

Retorna el escenario que en ese momento está cargado en el dron. En el caso de que la llamada sea no bloqueante, llamará a la función de callback pasándole el escenario. El formato del escenario es el indicado en la descripción del método `setScenario`.